

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

EXTENDING THE COMPUTER-AIDED SOFTWARE
EVOLUTION SYSTEM (CASES) WITH QUALITY FUNCTION
DEPLOYMENT (QFD)

by

Arthur B. Clomera

June 2003

Thesis Advisor:
Thesis Co-Advisor:

Man-Tak Shing
Joseph F. Puett III

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2003	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Extending the Computer-Aided Software Evolution System (CASES) with Quality Function Deployment (QFD).			5. FUNDING NUMBERS	
6. AUTHOR(S) Clomera, Arthur B.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) This thesis extends the Computer Aided Software Evolution System (CASES) with Quality Function Deployment (QFD) to enhance dependency traceability (type and degree) between software development artifacts. Embedding Quality Function Deployment (QFD) in the Relational Hypergraph Software Evolution Model to prototype a Holistic Framework for Software Engineering (HFSE) is the major task achieved by this thesis. CASES is implemented by using Java Development Kit (JDK) 1.3.1 and an open software architecture. The primary contributions of this research include: 1) Embedding QFD into CASES to record and track artifact dependencies, 2) Providing engineering views of QFD dependencies, and 3) Providing a stakeholder Graphical User Interface (GUI) to define and manage any software evolution process. These major contributions allow a software engineer to: 1) Input, modify, and analyze dependency characteristics between software artifacts within a QFD framework; 2) Make decisions based upon views of dependency information; and 3) Design a custom software evolution model through the use of a GUI.				
14. SUBJECT TERMS Software Engineering, Software Evolution, and Integrated Software Development Environments, Software Quality Function Deployment.			15. NUMBER OF PAGES 469	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**EXTENDING THE COMPUTER-AIDED SOFTWARE EVOLUTION SYSTEM
(CASES) WITH QUALITY FUNCTION DEPLOYMENT (QFD)**

Arthur B. Clomera
Major, United States Army
B.S., University of California, Davis, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
June 2003**

Author: Arthur B. Clomera

Approved by: Man-Tak Shing
Thesis Advisor

Joseph F. Puett III
Thesis Co-Advisor

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis extends the Computer Aided Software Evolution System (CASES) with Quality Function Deployment (QFD) to enhance dependency traceability (type and degree) between software development artifacts. Embedding Quality Function Deployment (QFD) in the Relational Hypergraph Software Evolution Model to prototype a Holistic Framework for Software Engineering (HFSE) is the major task achieved by this thesis. CASES is implemented by using Java Development Kit (JDK) 1.3.1 and an open software architecture. The primary contributions of this research include: 1) Embedding QFD into CASES to record and track artifact dependencies, 2) Providing engineering views of QFD dependencies, and 3) Providing a stakeholder Graphical User Interface (GUI) to define and manage any software evolution process.

These major contributions allow a software engineer to: 1) Input, modify, and analyze dependency characteristics between software artifacts within a QFD framework; 2) Make decisions based upon views of dependency information; and 3) Design a custom software evolution model through the use of a GUI.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	MOTIVATION	1
1.	Holistic Framework for Software Engineering (HFSE) [PUET02, 03]	1
2.	Software Evolution [HARN99a]	2
3.	Artifacts in Software Evolution	3
4.	CASES 1.1[LEHC99].....	5
5.	Needed Enhancements.....	6
B.	METHODOLOGY	8
C.	CONTRIBUTIONS.....	10
D.	ORGANIZATION OF THESIS	11
E.	SUMMARY	11
II.	PREVIOUS WORK.....	13
A.	HOLISTIC FRAMEWORK FOR SOFTWARE EVOLUTION (HFSE)	13
1.	Summary.....	13
2.	Concepts Useful to the Thesis	15
B.	RELATIONAL HYPERGRAPH SOFTWARE EVOLUTION MODEL	16
1.	Summary.....	16
2.	Concepts Useful to the Thesis	16
a.	<i>Improving Generality and Extensibility.....</i>	<i>16</i>
b.	<i>Increased Dependency Richness.</i>	<i>17</i>
C.	COMPUTER-AIDED SOFTWARE EVOLUTION SYSTEM (CASES).....	17
1.	Summary.....	17
2.	Concepts Useful in the Thesis	21
D.	QUALITY FUNCTION DEPLOYMENT (QFD).....	22
1.	Basic QFD [CLAU88]	22
a.	<i>Summary.....</i>	<i>22</i>
b.	<i>Key Idea.....</i>	<i>22</i>
c.	<i>Concepts Useful in the Thesis</i>	<i>22</i>
2.	House of Quality (HOQ) [HAUS88].....	23
a.	<i>Summary.....</i>	<i>23</i>
b.	<i>Key Idea.....</i>	<i>25</i>
c.	<i>Concepts Useful in the Thesis</i>	<i>26</i>
3.	Software Quality Function Deployment [ZULT92].....	27
a.	<i>Summary.....</i>	<i>27</i>
b.	<i>Key Ideas</i>	<i>29</i>
c.	<i>Concepts Useful in the Thesis</i>	<i>31</i>

E.	RATIONAL PROJECTCONSOLE [RATI01]	32
1.	Summary.....	32
2.	Key Ideas.....	33
3.	Concepts Useful in the Thesis	34
F.	CHAPTER SUMMARY.....	35
III.	QFD DEPENDENCY RELATIONSHIPS	37
A.	THE HOUSE OF QUALITY (HOQ).....	37
1.	Introduction.....	37
2.	Creating Models	37
3.	Creating Dependencies	38
B.	CALCULATIONS	39
1.	Deployment Equations	39
2.	Downstream Calculations	41
3.	Downstream Calculations Example	42
4.	Upstream Calculations	43
5.	Upstream Calculations Example	45
C.	THE ROOF.....	45
1.	General.....	45
2.	Roof Example	46
D.	USER-DEFINED VIEWS	47
1.	Dependency Threshold	47
2.	Dependency Threshold Example	48
3.	Component Trace.....	50
4.	Component Trace Example	51
E.	CASES AND OTHER TOOLS.....	54
F.	CHAPTER SUMMARY.....	56
IV.	CASES EXTENSIONS.....	57
A.	INTRODUCTION.....	57
B.	CASES 1.1 ARCHITECTURE	57
C.	CASES 2.0 ARCHITECTURE	60
1.	Layered Package Architecture	60
a.	<i>Subsystem Layer</i>	61
b.	<i>Message Layer</i>	66
c.	<i>Responsibility Layer</i>	69
2.	Patterns	70
D.	DATA IMPORT	71
E.	LIMITATIONS OF THE CURRENT IMPLEMENTATION	72
F.	SUMMARY	74
V.	CONCLUSION	75
A.	SUMMARY	75
B.	CONTRIBUTIONS.....	76
C.	FUTURE DIRECTIONS.....	76
1.	Technical.....	77
2.	Conceptual	77

D.	CONCLUDING THOUGHTS	78
APPENDIX A.	CASES 2.0 TUTORIAL	81
A.	INSTALLATION INSTRUCTIONS	81
B.	MINIMUM SYSTEM REQUIREMENTS	82
C.	HELLO WORLD EXAMPLE.....	82
1.	Creating or Loading a Project.....	82
2.	Creating a Project Schema.....	84
3.	Creating Step Versions	86
4.	Creating Dependencies	89
5.	Importing Data.....	90
6.	Deploying Dependencies	93
7.	Viewing QFD Matrices	94
8.	Engineering Views	96
9.	SPIDER Functionality	98
10.	Calling Tools.....	105
11.	Managing Personnel Data	106
12.	Job Scheduling	107
13.	Deleting a Project.....	108
APPENDIX B.	CASES SOURCE CODE	109
A.	CASES PACKAGE.....	109
1.	Cases.CasesFrame.....	109
2.	Cases.CasesTitle	123
3.	Cases.ComponentContentFrame.....	124
4.	Cases.ComponentType	135
5.	Cases.ConnectionLinksFrame	138
6.	Cases.DecomposeListDialog.....	145
7.	Cases.DeleteDialog.....	149
8.	Cases.Dependency	154
9.	Cases.DependencyType	155
10.	Cases.EHL	157
11.	Cases.EditDecomposeFrame.....	158
12.	Cases.JobScheduleMenu	164
13.	Cases.ListDialog.....	167
14.	Cases.Personnel	171
15.	Cases.PersonnelFrame.....	176
16.	Cases.ProjectMenu	184
17.	Cases.ProjectSchemaFrame.....	187
18.	Cases.ReviewComponentContentDialog	210
19.	Cases.SkillTableFrame	214
20.	Cases.StepContent	218
21.	Cases.StepContentFrame	224
22.	Cases.StepType.....	236
23.	Cases.TraceFrame	238
24.	Cases.VersionControl	251
B.	CASES.GUI PACKAGE	253

	1.	Cases.GUI.CasesToolBar	253
	2.	Cases.GUI.DepAttributes	258
	3.	Cases.GUI.FileSystemModel	264
	4.	Cases.GUI.FileTreeModel	268
	5.	Cases.GUI.GraphPanel	269
	6.	Cases.GUI.GraphPanelFlag	285
	7.	Cases.GUI.Popup	287
	8.	Cases.GUI.SpiderMenu	290
	9.	Cases.GUI.ToolsMenu	292
C.		CASES.GUI.AVC PACKAGE	296
	1.	Cases.GUI.AVC.AVCCreateStepFrame	296
	2.	Cases.GUI.AVC.AVCMenu	304
	3.	Cases.GUI.AVC.AVCMergingFrame	306
	4.	Cases.GUI.AVC.AVCOpenStepFrame	314
	5.	Cases.GUI.AVC.AVCSplittingFrame	320
D.		CASES.GUI.CALENDARDIALOG PACKAGE	329
	1.	Cases.GUI.CalendarDialog.DateButton	329
	2.	Cases.GUI.CalendarDialog.DateChooser	330
E.		CASES.GUI.PSF PACKAGE	337
	1.	Cases.GUI.PSF.ComponentTypePanel	337
	2.	Cases.GUI.PSF.DependencyTypePanel	338
	3.	Cases.GUI.PSF.EHLTypePanel	340
	4.	Cases.GUI.PSF.StepTypePanel	343
F.		CASES.GUI.UTIL PACKAGE	344
	1.	Cases.GUI.util.NextVersion	344
	2.	Cases.GUI.util.PreviousVersion	345
	3.	Cases.GUI.util.Tools	346
G.		CASES.INTERFACES PACKAGE	347
	1.	Cases.Interfaces.I_AVC	347
	2.	Cases.Interfaces.I_AVCOpenStep	347
	3.	Cases.Interfaces.I_Cases	348
	4.	Cases.Interfaces.I_ComponentContent	349
	5.	Cases.Interfaces.I_EditDecompose	349
	6.	Cases.Interfaces.I_Personnel	350
	7.	Cases.Interfaces.I_ProjectSchema	350
	8.	Cases.Interfaces.I_StepContent	351
	9.	Cases.Interfaces.I_Trace	351
H.		CASE.QFD PACKAGE	352
	1.	Cases.QFD.CalcMatrix	352
	2.	Cases.QFD.CombinedHouseMatrix	354
	3.	Cases.QFD.ComponentQFD	361
	4.	Cases.QFD.HouseMatrix	362
	5.	Cases.QFD.MyDefaultTableModel	371
	6.	Cases.QFD.StepQFD	372
	7.	Cases.QFD.TraceDownstream	373

8.	Cases.QFD. TraceDownstreamView	374
9.	Cases.QFD. TraceUpstream	376
10.	Cases.QFD.TraceUpstreamView	377
11.	Cases.QFD.TraceViewGUI	379
12.	Cases.QFD.UpdateCombinedOrigin.....	381
13.	Cases.QFD.UpdateOrigin.....	383
14.	Cases.QFD.UserDefinedView	385
15.	Cases.QFD.UserDefinedViewGUI.....	388
I.	CASES.QFD.UTIL PACKAGE	391
1.	Cases.QFD.util.AVCTools.....	391
2.	Cases.QFD.util.ExcelCSVLexer	392
3.	Cases.QFD.util.GetDoubleValue	404
4.	Cases.QFD.util.MatrixCalculator	405
5.	Cases.QFD.util.ObjectFileOperations	406
J.	JOBSCHEDULE PACKAGE.....	408
1.	JobSchedule.JAboutDialog	408
2.	JobSchedule.JDialog_jobskill	410
3.	JobSchedule.JDialog_message.....	412
4.	JobSchedule.JDialog_message1	414
5.	JobSchedule.JDialog_weight.....	416
6.	JobSchedule.JDialog_weit.....	418
7.	JobSchedule.JDialog_assignjob	420
8.	JobSchedule.JDialog_manager.....	430
9.	JobSchedule.Predecessor.....	442
10.	JobSchedule.Result	442
11.	JobSchedule.Weight.....	443
12.	JobSchedule.Work_schedul	443
	LIST OF REFERENCES.....	445
	INITIAL DISTRIBUTION LIST	447

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1	Typical Software Development Process Interaction (from [PUET03]).....	2
Figure 2	Software Evolution Model (from [HARN99a]).....	3
Figure 3	CASES Stakeholders (from [LEHC99]).....	4
Figure 4	CASES 1.1 New Project Screenshot.....	5
Figure 5	CASES 1.1 Project Schema Creation Dialog	6
Figure 6	CASES 2.0 Completed Project Schema.....	7
Figure 7	Research Methodology	8
Figure 8	Overall System Integration Use Case	9
Figure 9	Holistic Model of Software Process Interaction	13
Figure 10	Object-Oriented Design Pattern Model-View-Controller.....	15
Figure 11	SPIDER(1 and 2) (after [LEHC99])	18
Figure 12	CASES 1.1 Elided Class Diagram (after [LEHC99]).....	20
Figure 13	First Level QFD Matrix -- The "House of Quality" (after [PUET03]).....	24
Figure 14	Kano Model (after [ZULT92]).....	28
Figure 15	Example of a QFD Matrix Deployment (from [PUET03])	29
Figure 16	QFD Model drawn as a Process Diagram.....	32
Figure 17	ProjectConsole Dashboard (from [RATI01])	34
Figure 18	Project Schema Creation.....	38
Figure 19	Dependency Dialog.....	39
Figure 20	QFD Matrix for Risk Deployment Example.....	40
Figure 21	Downstream Dependency Calculations Code.....	42
Figure 22	QFD Matrix for Requirements Risk Deployment Downstream Calculation...43	
Figure 23	Upstream Dependency Calculation Code	44
Figure 24	QFD Matrix for Specification Risk Deployment Upstream Calculation.....	45
Figure 25	Dependency Network Diagram.....	46
Figure 26	HOQ Roof.....	47
Figure 27	Dependency Threshold Example	48
Figure 28	Dependency Threshold Dialog Window.....	49
Figure 29	Dependency Threshold Result	49
Figure 30	Dependency Threshold Calculations Code	50
Figure 31	Weighted Digraph Example.....	51
Figure 32	Component Trace Dialog Window	52
Figure 33	Component Trace Example.....	52
Figure 34	Component Trace Results	53
Figure 35	Component Trace Calculations Code	53
Figure 36	Import Dependency Value Confirmation.....	54
Figure 37	Dependency List Dialog	55
Figure 38	Column Header List Dialog.....	56
Figure 39	CASES 1.1 Class Diagram (from [LEHC99]).....	58
Figure 40	Composition of CASES 2.0 ProjectSchemaFrame	60
Figure 41	Cases Package Diagram.....	61

Figure 42	Cases.GUI Package Diagram.....	62
Figure 43	Cases.QFD Package Diagram.....	63
Figure 44	Aggregation of Types	64
Figure 45	Individual Calculation Sequence Diagram	65
Figure 46	Cases.JobSchedule Package Diagram.....	66
Figure 47	Cases.Interfaces Package Diagram	67
Figure 48	CASES Application MVC Architecture	70
Figure 49	UML for Current Import Data Architecture	71
Figure 50	Recommended Import Data Architecture	72
Figure 51	Trace Pseudo Code	73
Figure 52	Extract from CasesTitle of GraphPanel Limitations.....	74
Figure 53	Create Project.....	83
Figure 54	HelloWorld Project.....	83
Figure 55	Open Project.....	84
Figure 56	Popup Menu.....	85
Figure 57	Component Project Schema.....	86
Figure 58	Automated Version Control.....	86
Figure 59	Create Step Version	87
Figure 60	Open Step Version	88
Figure 61	Splitting Step Version.....	88
Figure 62	Merge Step Version	89
Figure 63	Dependency Dialog.....	90
Figure 64	Import CSV Files.....	91
Figure 65	Import Dependency Values.....	91
Figure 66	Open HelloWorld Reqts 1.1 CSV File.....	92
Figure 67	Dependency List	92
Figure 68	Column Header List.....	93
Figure 69	QFD Matrix for Risk Deployment Example.....	94
Figure 70	Open QFD Matrix Risk v.1.1.....	94
Figure 71	House of Quality Risk v.1.1.....	95
Figure 72	Roof Risk v1.1	95
Figure 73	HOQ Roof.....	96
Figure 74	Dependency Threshold View.....	97
Figure 75	Column or Row Trace.....	97
Figure 76	Trace View.....	98
Figure 77	SPIDER.....	99
Figure 78	Directory Tree.....	99
Figure 79	Edit SPIDER.....	100
Figure 80	Decompose SPIDER.....	100
Figure 81	SPIDER Component Content	101
Figure 82	Component Content Editor	102
Figure 83	SPIDER Step Content.....	103
Figure 84	Skill List.....	103
Figure 85	Predecessor List	104
Figure 86	Calendar Dialog	105

Figure 87	SPIDER Trace.....	105
Figure 88	CASES Tools	106
Figure 89	Add Personnel Data	106
Figure 90	Edit Personnel Data.....	107
Figure 91	No Job or Stakeholder Assigned Error	107
Figure 92	Delete Project.....	108

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1	Steps of Software Evolution Process	19
Table 2	Components of Software Evolution Process.....	19
Table 3	Roof Dependency Direction	47
Table 4	Software Metric Description.....	59
Table 5	Basic Software Metrics	59
Table 6	CASES Interfaces (after [LEHC99])	68
Table 7	Image Package	69
Table 8	Data Package.....	69
Table 9	Responsibility Package Layer.....	70
Table 10	Minimum System Requirements.....	82

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to give thanks to the Lord for blessing my life with a loving family (Caroline, Jessica, and Zachary), precious opportunities, and a career defending the greatest country in the world. My life has been enriched through this experience at the Naval Postgraduate School.

I would like to thank LTC Joe Puett for his many hours of mentorship and inspiration. He has encouraged me to learn and achieve beyond my expectations. His dedication to excellence has benefited me tremendously and will provide energy throughout my life.

Additionally, I would like to thank Professors Man-Tak Shing and Richard Riehle for their professional assistance throughout my studies and research at the Naval Postgraduate School. They have been supportive during my research and instrumental during my studies. They have provided me with the academic assistance making all this possible.

To all these individuals and others not mentioned, I thank you from the sincerity of my heart.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

1. Holistic Framework for Software Engineering (HFSE) [PUET02, 03]

One means of constructing quality software is to leverage the traceability and dependencies of software evolution artifacts through the stages of requirements, specifications, architecture, design, implementation, and testing. A capability to track relationships and to relate them when change occurs forms a key holistic thread throughout many modern high-assurance software evolution processes, particularly in mission-critical activities, business financial transactions, and life-critical medical applications. This important issue cannot be solved without a holistic methodology that allows the software development tools to work together more efficiently and effectively within a common framework. Until industry breaches this obstacle, there will continue to be problems with the software development quality, schedule, budget, complexity, and longevity.

This thesis provides supporting work for the implementation of a “Holistic Framework for Software Engineering (HFSE)” [PUET02, 03]. The HFSE establishes mechanisms by which existing software development tools and models can work together with greater efficiency and effectiveness. Volumes of research have been conducted to understand and improve the individual aspects of software development, but comparatively little research has been done on holistic models of how these various threads and processes could interact. A software engineer needs an efficient and effective interface between different development models and tools in the software development process (see Figure 1) in order to manage the complexity of large system development. The introduction of a software development support tool that forms the interface between a software engineer and the development models and tools that the engineer relies on provides such a basis. Embedding QFD in the Relational Hypergraph Software Evolution Model, prototyping the HFSE, and applying the HFSE to a selected set of tools are major tasks achieved by this thesis, which provide confirming evidence of the feasibility of the HFSE.

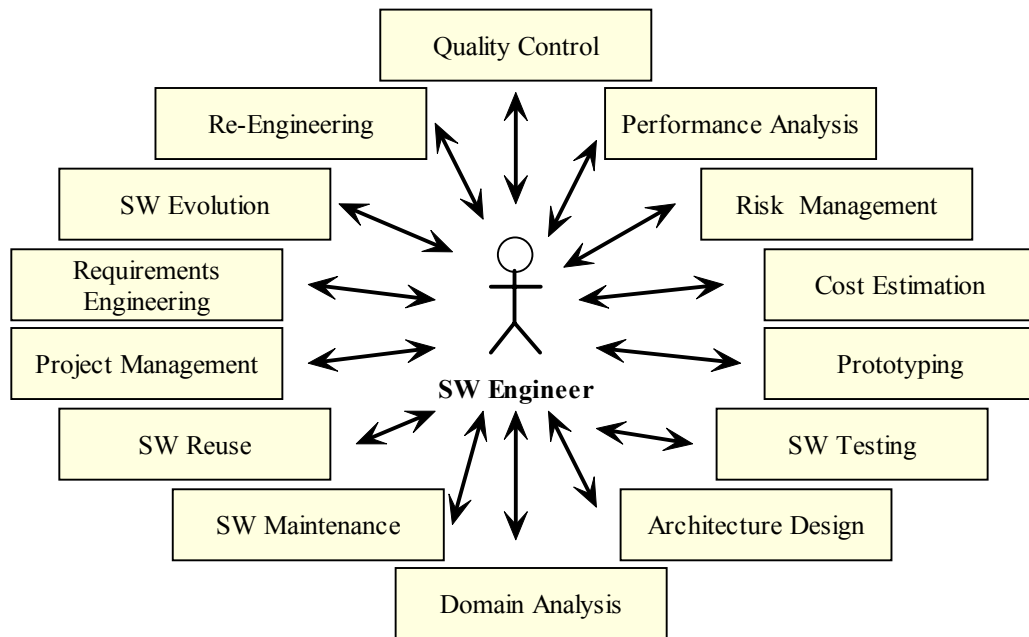


Figure 1 Typical Software Development Process Interaction (from [PUET03])

A holistic framework must be flexible enough to work with any development tool or model. It must be capable of recording relationship information between artifacts; but more importantly, it must assist a software engineer as a decision support tool with a solid, rigorous, and repeatable methodology.

2. Software Evolution [HARN99a]

Software evolution processes are the set of activities, changes, functions, and byproducts (both technical and managerial) that ensure software will continue to meet organizational and business objectives in a cost effective manner. Software evolution is concerned with corrections, improvements, and enhancements of software and the ability to maintain positive control of all software artifacts over the entire life cycle of the software development.

The Relational Hypergraph Model of software evolution [HARN99a] originated from the Issue-Based Information System (IBIS) model [CONK88]. The model relies on eight types of top-level steps (illustrated by squares in Figure 2) and eight types of top-level components (illustrated by circles in Figure 2). This particular model was rigidly

adhered to in previous work [HARN99a] [LEHC99] and unfortunately does not necessarily model the software processes actually used by industry. Preferably, the software evolution process model should be adaptable to any development methodology that a particular software development team uses.

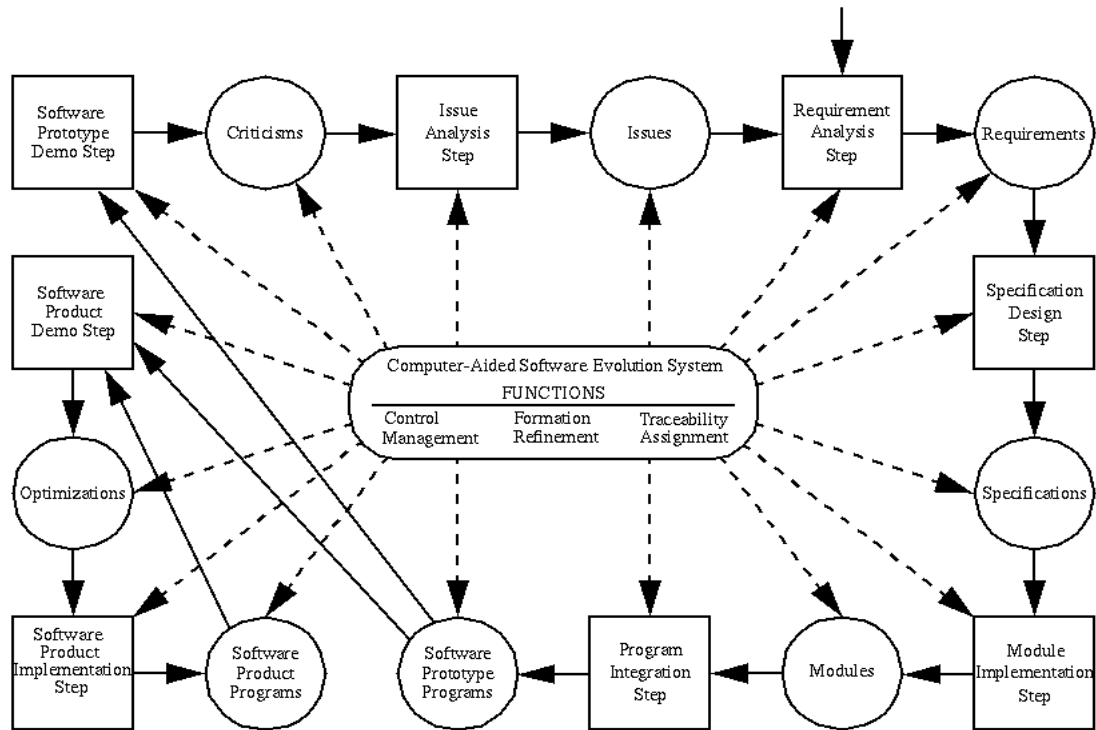


Figure 2 Software Evolution Model (from [HARN99a])

3. Artifacts in Software Evolution

The artifacts in the software evolution process model (see Figure 2) are called software evolution objects [HARN99a]. The evolution objects in the IBIS model include software evolution steps (8 steps) and components (8 components). These objects are monitored and manipulated with the java-based program known as the Computer Aided Software Evolution System (CASES) [LEHC99]. CASES is discussed in detail in Chapter II section C.

The eight types of steps in CASES include: requirements analysis, specification design, module implementation, program integration, software product demo, software product implementation, software prototype demo, and issue analysis. The step attributes

within CASES include: version and variation number, priority, status, skills and levels, security level, predecessor, temporal information, etc. These attributes give the stakeholder utilizing CASES the capability to track small and large changes during the evolution of the software product (see Figure 3).

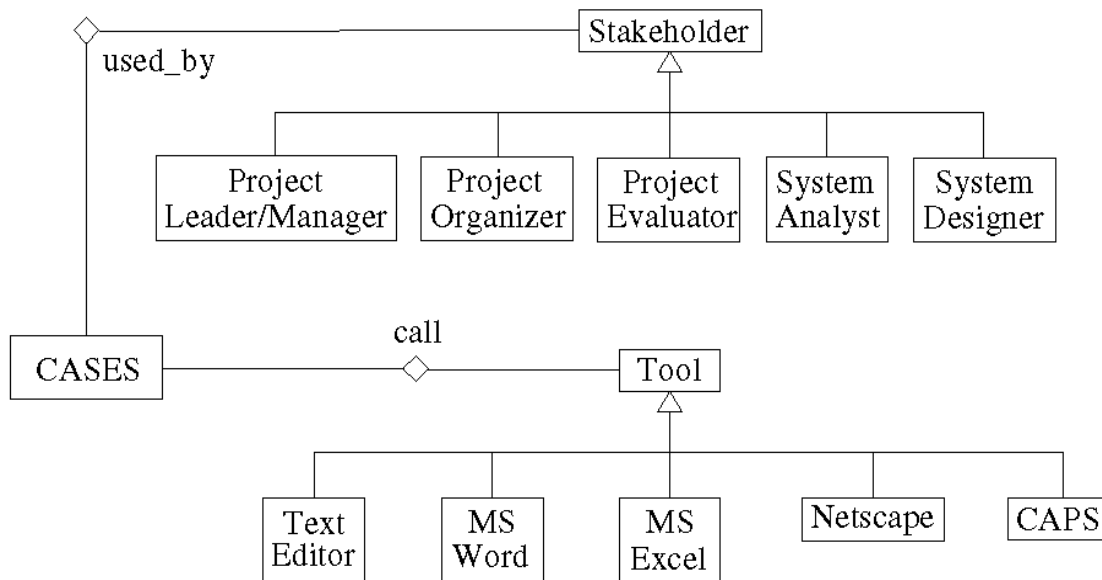


Figure 3 CASES Stakeholders (from [LEHC99])

The eight types of components within CASES are the following: requirements, specifications, modules, software prototype programs, optimizations, software product programs, criticisms, and issues. The component attributes include the following: hypertext, software code, pictures, charts, movies, and data from other software engineering tools (requirements engineering tools, prototyping tools, etc.). These attributes give the stakeholder utilizing CASES the capability to track the software artifacts created by particular steps. Stakeholders can use CASES to invoke software engineering tools such as a web browser, office tools, and software development tools. This capability improves communication between stakeholders and reduces time to gather information about the software system.

4. CASES 1.1[LEHC99]

Hanh Le [LEHC99] developed the automated software evolution tool known as the Computer-Aided Software Evolution System (CASES 1.1) in support of the Relational Hypergraph (RH) Model of software evolution (see Figure 4).



Figure 4 CASES 1.1 New Project Screenshot

The objective of her study was to design and implement CASES to demonstrate the research performed by [HARN99a]. CASES was developed using object-oriented tools, Java development kit (JDK) 1.1.7 and Swing 1.0.3. under the Webgain Visual Café version 3.0 environment. CASES 1.1 was the first software evolution tool to support the practicality of the RH model. The usability of CASES was constrained by a menu driven and text based user input software model (see Figure 5).

Project Schema

Step Type | Component Type | Evolution Process | Dependency

Project Label: Test5

Step Type ID:

Step Type Name:

Existing Step Types:

Step Type Description:

Add Edit Delete Clear Save Finish

Figure 5 CASES 1.1 Project Schema Creation Dialog

A project schema must be defined in order to apply the automated version control, Step Processed In Different Entrance Relationships (SPIDERS), tools, and job schedules to a software evolution project. This textual view of a project schema degrades the pragmatic adoption of the CASES application. Step and Component identifications must be meticulously formatted by the user in order for the program to execute correctly. Additionally, the project schema is further complicated by the requirement for the user to textually define evolution processes (the flow of components and steps) and dependencies (primary and secondary input).

5. Needed Enhancements

CASES 1.1 does have many useful features however, which include version and variance control, support of the RH model, and logical links to component artifacts. But, the issues discussed in the prior section, the need for a holistic framework for software evolution, and a new object-oriented design, warrant improving and extending CASES. CASES 2.0 was developed to meet those needed enhancements without reducing the current capabilities of CASES 1.1. These enhancements are discussed in greater detail in

Chapter III and IV of this thesis. Readers interested in additional details of Holistic Framework for Software Evolution are referred to [PUET03].

CASES 2.0 has an interactive GUI by which the user can design and populate a software evolution project schema (see Figure 6). This GUI hides the implementation from the user; therefore, the user does not have to be concerned with artifact identification, evolution processes definition, and primary and secondary input definitions. CASESv2 adds new capability through embedding QFD into the project schema. Embedding QFD is a decision support tool, which provides a holistic framework for modeling any project schema.

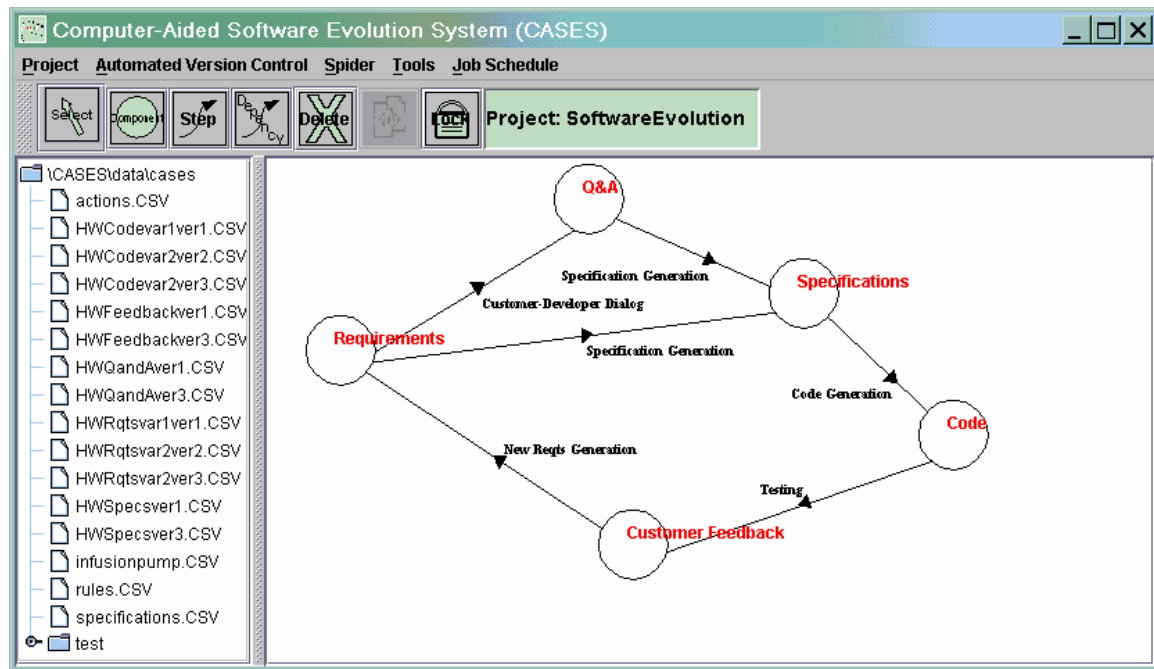


Figure 6 CASES 2.0 Completed Project Schema

CASES 2.0 consists of several packages, instead of a single package of thirty-five Java files. CASESv2 takes advantage of a three layer object-oriented design; packages are divided into three layers: subsystem, message, and responsibility. Further software architectural details are provided in Chapter IV of this thesis. Together, these architectural improvements give CASES a logical structure for managing project cohesion.

B. METHODOLOGY

The enhanced software evolution tool developed in support of this thesis (CASES v2.0) was built using object-oriented techniques with JDK 1.3.1. An iterative approach was used (see Figure 7). This approach consisted of four life cycle phases: inception, elaboration, construction, and transition. During inception, the focus was on understanding and analyzing the source code for CASES 1.1, identifying shortcomings and defects, determining a scope for this thesis, and researching the feasibility of an implementation.

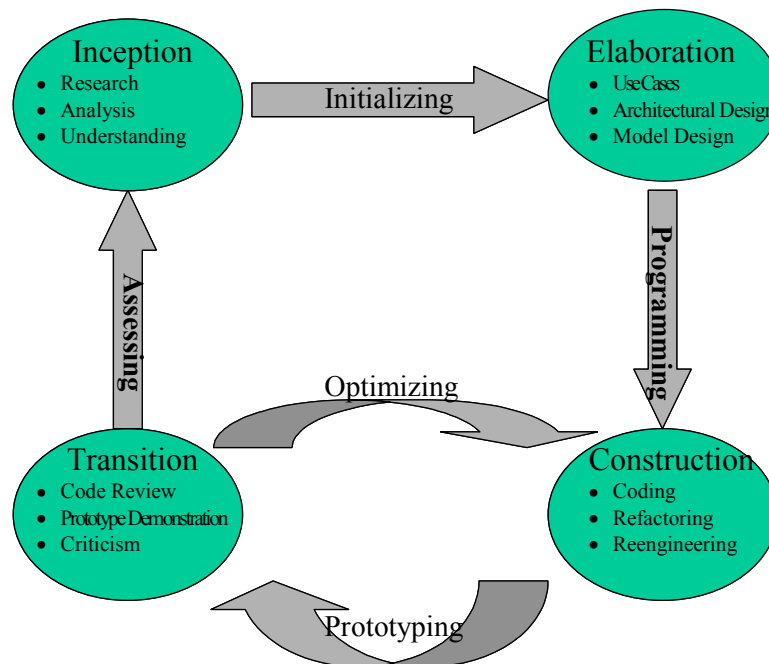


Figure 7 Research Methodology

During elaboration, the requirements for the modifications were defined, an initial executable architecture was established with UML Use Cases, and the architecture was modeled with TogetherSoft Corporation's Together 6. During construction, an open software architecture and design was implemented in Together 6 and coding was completed. Previous source code was reengineered and/or refactored to comply with an open software architecture. Finally, a transition phase was conducted to review the source code, conduct unit and validation testing, and to demonstrate the capabilities to

and receive criticism from the stakeholders. As needed changes were identified, the approach was to undertake additional development iterations until a satisfactory product was achieved. On limited occasions, an assessment of the model led to revisiting the inception phase to modify the architecture.

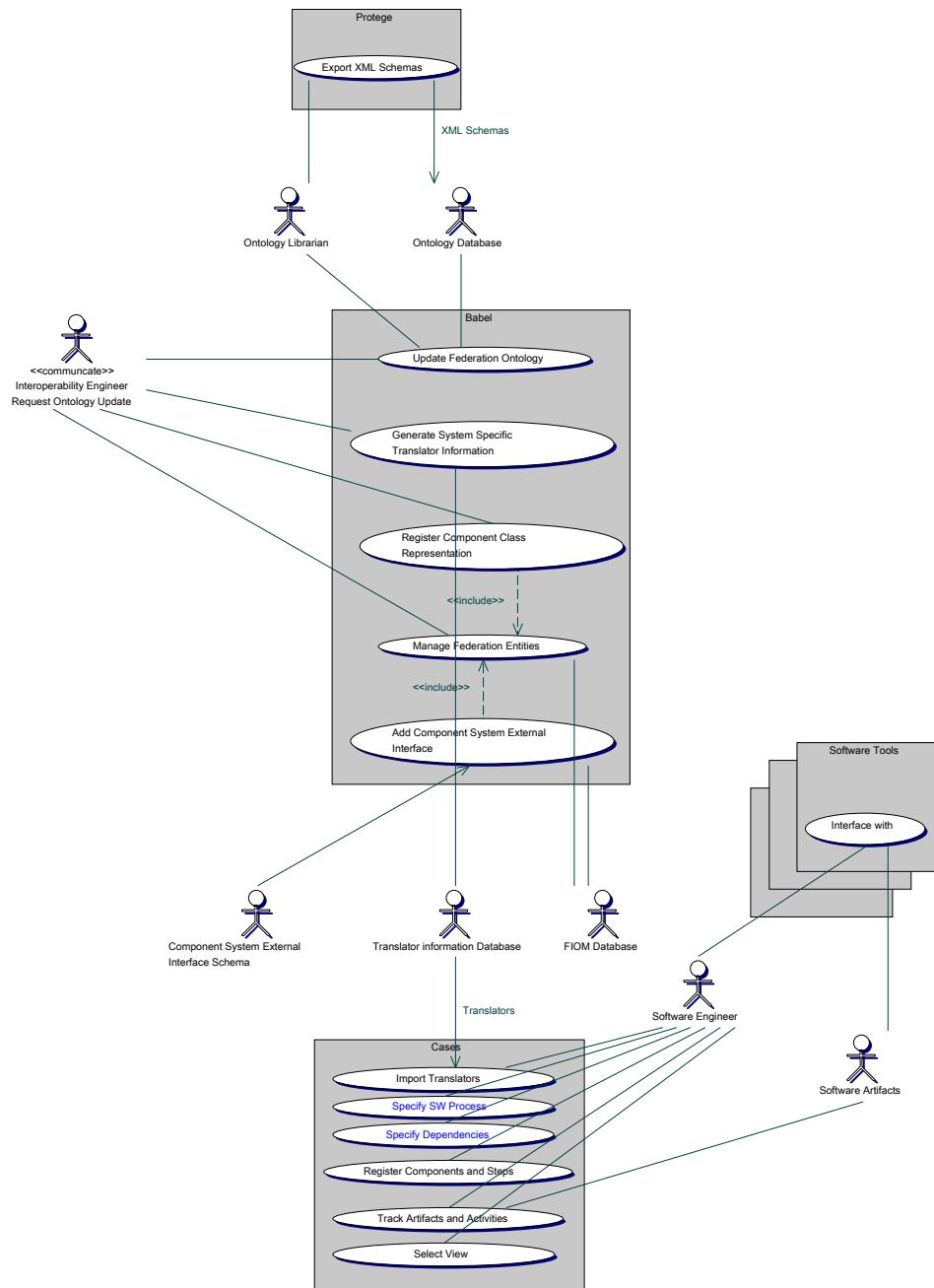


Figure 8 Overall System Integration Use Case

The overall system integration use case (see Figure 8) provides a graphical representation of the customer's vision of CASES' interoperability with other actors (the translator information database and software engineers) and systems (software tools). Figure 8 uses standard UML notation where the solid line represents an association, a solid line with an arrowhead represents an association with its direction to be read, and a dashed line with an arrowhead represents a dependency from client to supplier. The complete use case definition is presented in [PUET03] and provides some goals for future developers of CASES. Future goals for CASES include: interoperability with the Object-Oriented Model for Interoperability (OOMI) IDE (also known as Babel) through a translator information database, capability to track artifacts and activities from software tools, and interface with an ontology librarian to the external ontology support system (e.g. Protégé).

CASES 2.0 was designed using an open, modular architecture; therefore all dependencies on Webgain Visual Café version 3.0 were eliminated from the [LEHC99] version of CASES. The new release of CASES has increased portability, maintainability, and understandability. CASES can interface with Microsoft Notepad, Word, Excel, and Internet Explorer, the prototyping suite SEATools (Software Engineering Automation Tools), and a requirements tool (Rational Rose Requisite Pro).

C. CONTRIBUTIONS

The most important contribution to the field of Software Engineering that this thesis provides is to:

- Embed QFD within the Relational Hypergraph Software Evolution Model,
- Provide engineering views of QFD dependencies, and
- Provide a stakeholder GUI for defining software development processes.

Additional contributions include:

- Designing and implementing a modular software architecture, and
- Eliminating the dependency on the Visual Café development environment.

D. ORGANIZATION OF THESIS

In Chapter II, previous work is presented and discussed. In Chapter III, Quality Function Deployment (QFD) dependency relationships are presented and the specifics on how calculations are performed for the “house of quality” are derived. In Chapter IV, the objects and functions, which were created to extend CASES, are illustrated. Additionally, modifications to original code and existing limitations are identified. Chapter V contains the conclusions and recommendations for future work.

E. SUMMARY

The objective of this thesis is to embed QFD into a software evolution model to assist software engineers with software evolution activities and provide decision support. This provides a holistic framework to control and manage the software evolution processes through QFD and provides an open, modular software architecture for future improvements.

The flexibility of this tool allows software engineers to interface with existing and future software development tools and models while providing a holistic framework to view and reason about dependency information. Such capability hopes to reduce total life cycle costs, improve software product quality, and reduce evolution timelines.

THIS PAGE INTENTIONALLY LEFT BLANK

II. PREVIOUS WORK

A. HOLISTIC FRAMEWORK FOR SOFTWARE EVOLUTION (HFSE)

1. Summary

Puett, in his PhD dissertation [PUET03], defines the Holistic Framework for Software Evolution (HFSE) as “a conceptual framework for establishing interoperability between software development tools as well as a methodology (with tool support) that assembles the necessary objects and interoperability constructs.” It can be viewed as “an abstract layer of activity that interacts with subordinate software development tools via middleware communications mechanisms” (see Figure 9). Figure 9 implies that the HFSE will provide the stakeholder(s) a generic view of the relationships between software development artifacts within a single software development tool context.

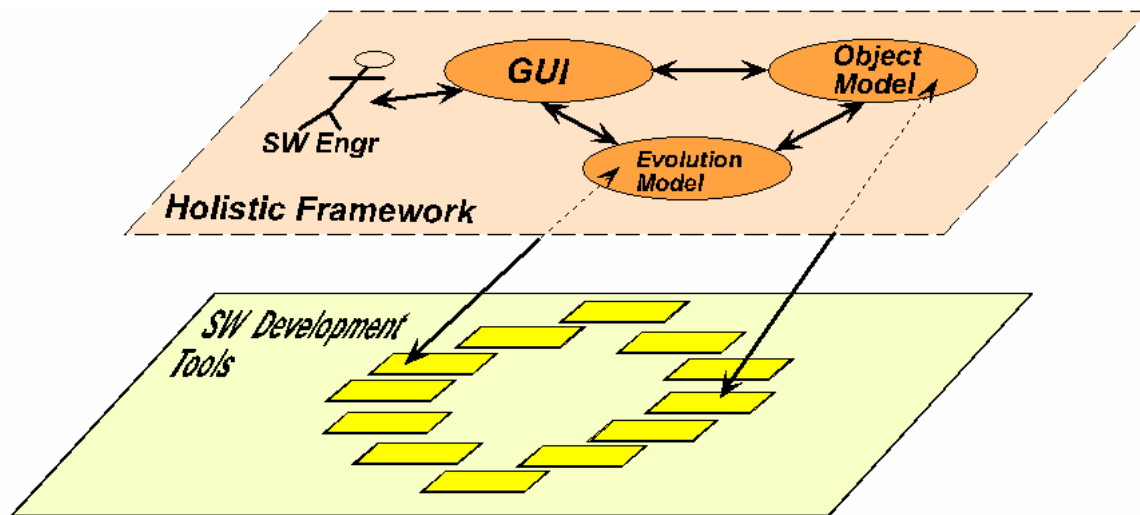


Figure 9 Holistic Model of Software Process Interaction

The Evolution and Object Models abstract data from software development tools and processes. [PUET03] identifies the following research considerations that must be addressed when establishing a higher-level holistic framework:

- Identify standards for representation and interpretation of information

- Establish the medium of communication
- Account for process order
- Provide missing data
- Account for ambiguity of inputs and outputs
- Account for conflict resolution between models
- Provide for extensibility

[PUET03] points out that an evolution interface is required to automatically deploy a range of artifact dependencies throughout the lifetime of a particular software project. The role of the interface is to extend the preexisting Software Evolution model [HARN99a] with QFD to introduce a continuum of dependencies between artifacts. This continuum, at the abstract layer, filters relevant dependencies from noisy data. The extensions provide improvements over the previous Software Evolution model [HARN99a] and improve the vertical, horizontal, and temporal dependency graph between software artifacts.

Additionally, [PUET03] explains that an ideal HFSE should have the following characteristics:

- Generic and non-proprietary
- Support common software development tools
- Software development process independence
- Adaptable to any problem domain
- Extensible to new technologies and attributes as required
- Improve the software product time to market
- Decrease cost of software development
- Improve software quality
- Intuitive and easy to use

While achieving all of these characteristics is a long-term goal for the HFSE, the tool support provided by CASESv2 addresses only some of these characteristics.

2. Concepts Useful to the Thesis

The key ideas in [PUET03], which are important to this thesis, are that the construction of HFSE is feasible, that HFSE artifacts can be described mathematically, and that the HFSE improves software tool interoperability by allowing information provided from different tools to be used together for new purposes. Rational ProjectConsole (discussed later) is a similar implementation example of this concept. It provides a central abstraction of many software tools' data, but in the case of ProjectConsole, the abstracted data are independent not holistic. An important concept within HFSE is that it improves software quality and software product time to market. This is accomplished through a holistic framework of software tools working coherently. [PUET03] provides the direct motivation for embedding QFD into CASES.

The CASES extensions implemented in this thesis provide the necessary tool support needed for establishing an initial HFSE. The abstraction of data of the Evolution and Object Models are implemented in CASES 2.0 with the Model-View-Controller object-oriented pattern (see Figure 10).

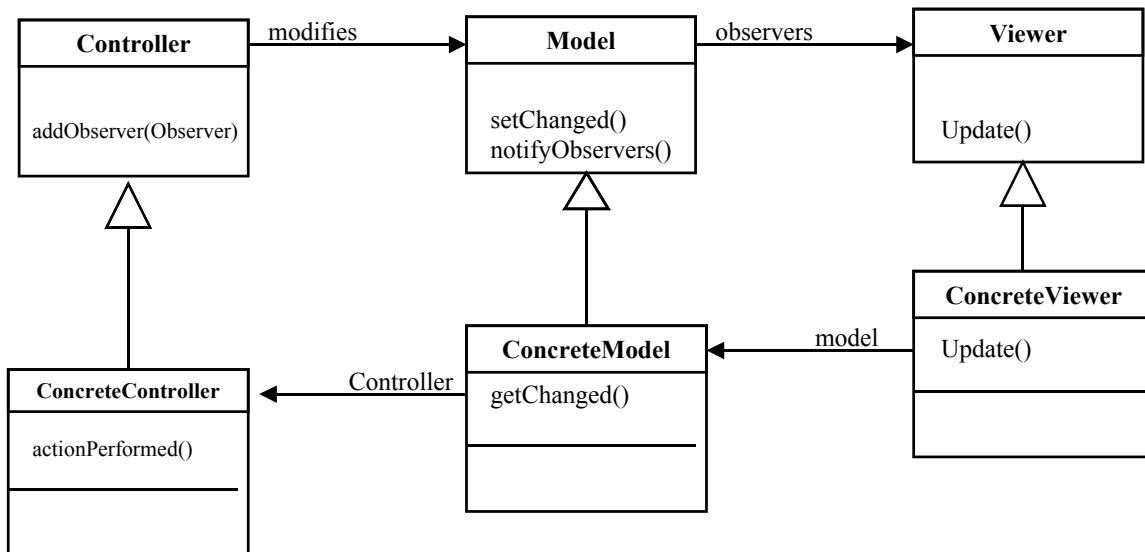


Figure 10 Object-Oriented Design Pattern Model-View-Controller

B. RELATIONAL HYPERGRAPH SOFTWARE EVOLUTION MODEL

1. Summary

The model of the software evolution process used in this thesis is based on the Relational-Hypergraph (RH) model and the Issue-Based Information System (IBIS) model. [HARN99a] defines a hypergraph model, evolutionary hypergraph, and relational hypergraph as:

A hypergraph model represents the evolution history and future plans for software development as a hypergraph. Hypergraphs generalize the usual notion of a directed graph by allowing hyperedges, which may have multiple output nodes and multiple input nodes.

An evolutionary hypergraph is a directed, labeled hypergraph that has been annotated with the attributes of the evolutionary components and steps of a software development process.

A relational hypergraph is an evolutionary hypergraph in which the dependency relationships between components and steps can have a hierarchy of specialized interpretations.

The RH model allows the development of tools to manage the activities (steps) and artifacts (components) in a software evolution project. CASES is an example of such a tool. A RH model is a set of vertices (or nodes) connected through a set of hyperedges. CASES implements a RH model through *serializable* objects (ComponentType and StepType) and configuration files (e.g. dependency.cfg, step.cfg, component.cfg, input.p, and input.s).

2. Concepts Useful to the Thesis

This thesis work focuses on extending the previous work of [LEHC99] and [HARN99a]. There are two primary extensions to CASES and the RH model which embedding QFD into CASES produces:

a. *Improving Generality and Extensibility.*

CASESv2 provides a GUI interface for users to build unique project schemas tailored specifically to their development process. CASESv2 hides the

implementation that frees the users from any single process model (specifically, the Evolutionary Prototyping model illustrated in Figure 2).

b. Increased Dependency Richness.

CASESv2 extends the dependency definitions within the RH model (e.g. primary-input driven and secondary-input driven). CASESv2 uses QFD as the framework for users to build and track customized dependency relationships (e.g. risk, safety, difficulty, priority, etc.) between artifacts (components).

Embedding QFD into the CASES' RH model is the major thrust of this thesis work. Additionally, QFD provides the tool support for the HFSE by allowing for and keeping track of a continuum of dependencies between objects.

C. COMPUTER-AIDED SOFTWARE EVOLUTION SYSTEM (CASES)

1. Summary

[LEHC99] developed an automated software evolution tool, the Computer-Aided Software Evolution System (CASES) that is based on the Relational Hypergraph (RH) Model of software evolution. The objective of her study was to design and implement CASES in support of [HARN99a]. CASES assists the software engineer in performing software evolution activities and allows the engineer to better control and manage the software evolution process. CASES provides automated assistance throughout software evolution and supports the practical development of large complex systems designed and built by physically distributed development teams. CASES provides five functions related to the activities of software evolution: 1) step refinement, 2) project evaluation, 3) constraint management, 4) personnel management, and 5) step management. Additionally, CASES provides five functions related to software evolution components: 1) component management, 2) component traceability, 3) configuration management, 4) dependency management, and 5) inference rule management. CASES 1.1 was developed using object-oriented tools, Java development kit (JDK) 1.1.7, and Swing 1.0.3. under the Visual Café version 3.0 environment. The work presented in [LEHC99] is divided into three major sections: the RH Model, CASES design, and CASES implementation.

The RH Model used in CASES is an evolutionary hypergraph in which the dependency relationships between components and steps can have a hierarchy of specialized interpretations. These can be used to refine the software evolution process model. In [LEHC99], there are two dependency relationships identified: primary and secondary. A *primary-input-driven* path is that path along an evolutionary hyperedge to an output node, where the input and output nodes are different versions of the same component.

A *secondary-input-driven* path is that path along a hyperedge from an input node to an output node, where the nodes are of different components (i.e. not different versions of the same component). A software evolution step with its input and output components form a Step Processed In Different Entrance Relationship (SPIDER) shown in Figure 11, which simplifies the dependency complexity of software evolution and helps to construct the relational hypergraph net. Each top-level, refined, and atomic level SPIDER is connected into a relational hypergraph net. An atomic level SPIDER is a minimal task unit that can be assigned to developers.

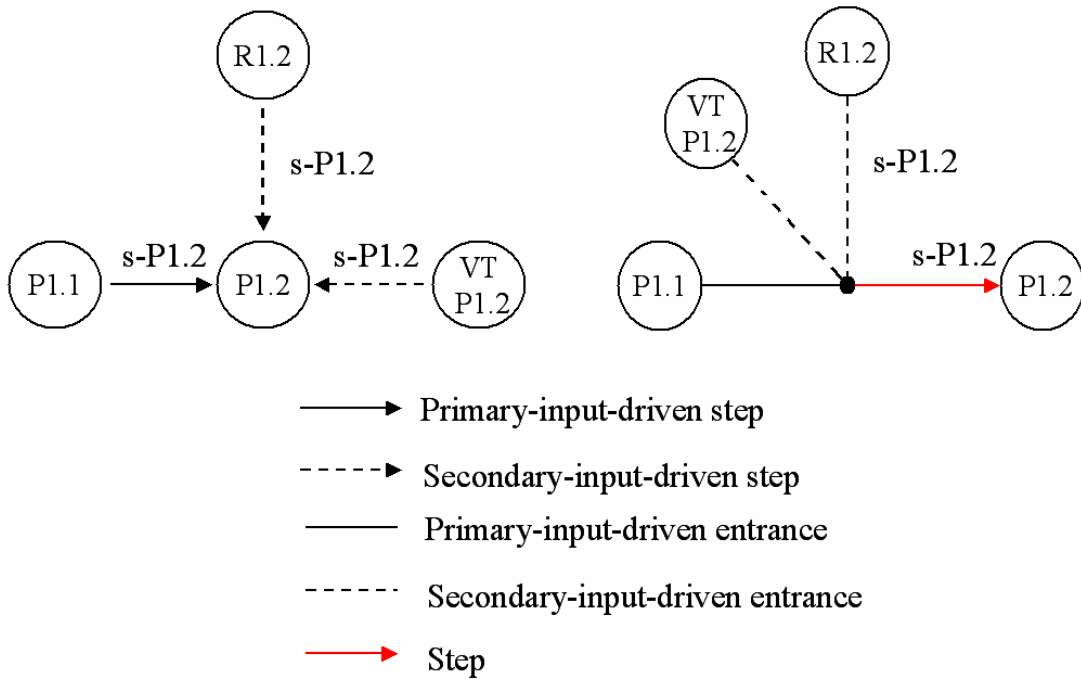


Figure 11 SPIDER(1 and 2) (after [LEHC99])

Additionally [HARN99b] provides the following definition of a relational hypergraph net:

A relational hypergraph net has two relational hypergraphs: a top-level relational hypergraph and a refined relational hypergraph. A complicated relational hypergraph can be transferred into a relational hypergraph net for simplifying the hypergraph structure. The top-level and the refined relational hypergraphs are respectively connected by top-level and refined SPIDERS (Step Processed in Different Entrance Relationships) that are formed by a specified step together with its input components and unique output component.

CASES allows the user to define steps and components tailored to a specific software development methodology; however, for illustrative purposes the process model described in [LEHC99] is the evolutionary process model. CASES manages and controls all of the activities that change a software system and the relationships among these steps and components. CASES is based upon the relationships of software evolution shown previously in Figure 2. Table 1 and Table 2 provide a legend for these steps and components as implemented in CASES.

Top-Level Steps	Symbol
Software prototype demo	s-C
Issue analysis	s-I
Requirement analysis	s-R
Specification design	s-S
Module implementation	s-M
Program integration	s-P
Software product demo	s-Pd
Software product implementation	s-O

Table 1 Steps of Software Evolution Process

Top-Level Components	Symbol
Criticisms	C
Issues	I
Requirements	R
Specifications	S
Modules	M
Software prototype programs	P
Optimizations	O
Software product programs	Pd

Table 2 Components of Software Evolution Process

CASES 1.1 is designed with thirty-five java files that are launched by the *CasesFrame.java* file. While the details of each class have been elided, Figure 12 illustrates the eighteen major classes and their associations (note: the eight minor classes and nine interfaces are not displayed to abstract out the central functionality). CASES 1.1 has a single package structure where most of the files are reusable through their class. All global variables are contained within the *CasesTitle.java* file, providing for the easy modification or updating of titles, filenames, constants, lists, and locations. The directory structure and the directory dependency and the file dependency rules are important factors in the design. This design uses the directory structure to store project information (such as step identifiers, critical files, version numbers, and SPIDER construction).

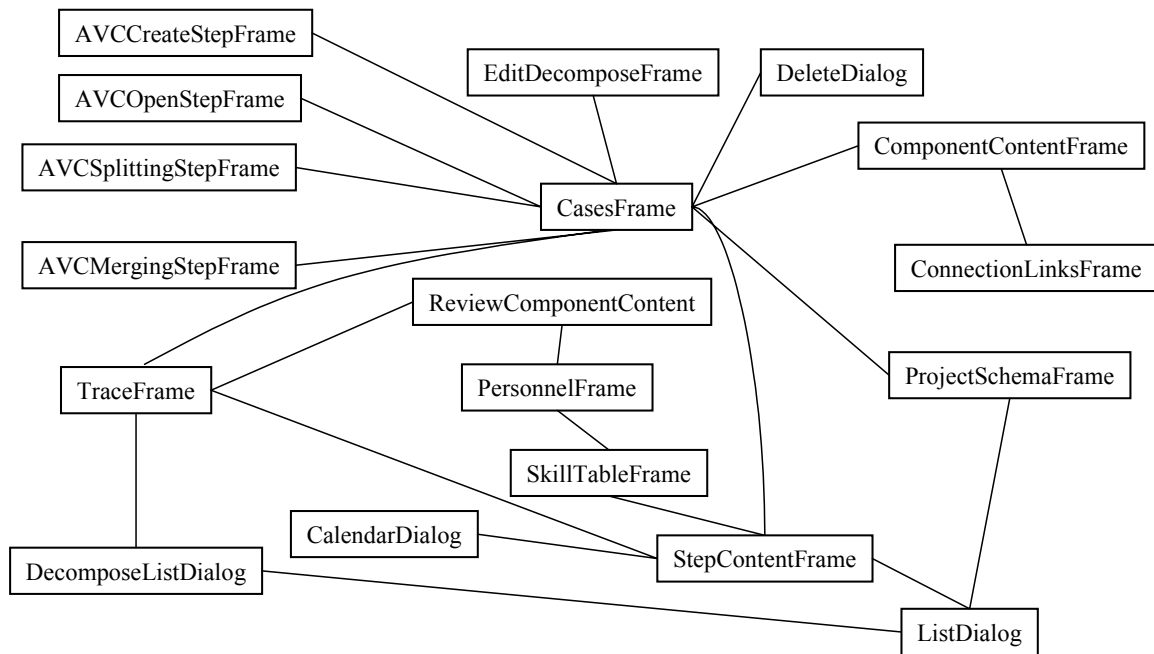


Figure 12 CASES 1.1 Elided Class Diagram (after [LEHC99])

CASES implementation is based on the multi-dimensional architecture of the RH model, which is embedded within the CASES directory structure. The file structure of CASES includes the CASES directory, project names, step identifiers and related files, version numbers, and SPIDER construction. CASES uses object serialization to write to and read back from input and output streams. There are seven object streams: 1) *step.cfg*, 2) *component.cfg*, 3) *loop.cfg*, 4) *current.vsn*, 5) *step.cnt*, 6) *dependency.cfg* and 7)

personnel objects. *Personnel* objects allow the user to describe and manage personnel attributes (ID, name, skill, security level, email, telephone, fax, and address). Additionally, the majority of Java files in CASES are applications, which are developed from the *Java Foundation Classes* (JFC) libraries (*JFrame*, *JDialog*, *JOptionPane*, *FileDialog*, and *JFileChooser*). *FileDialog* is the only application that is a direct subclass of the *java.awt* package. Many features within the applications in CASES 1.1 are provided through the Visual Café libraries.

2. Concepts Useful in the Thesis

The key ideas in [LEHC99], which are important to this thesis, are that automated support tools for software evolution are possible and that CASES 1.1 tracks dependency relationships between components and steps using a primary and secondary input dependency scheme. CASES 1.1 was designed for future researchers to continue to build upon and modify to make it an even more effective software evolution tool. Its object-oriented design provides for easier understandability, modification, and extensibility.

CASES 1.1 is the first version to support the practical implementation of the RH model. The objective of this thesis research is to extend Le's work [LEHC99] to make CASES a more effective software evolution tool by: 1) adding richness to the dependency relationships between software development components 2) updating the source code to the current version of the JDK and 3) removing references to Visual Café libraries. Object-oriented analysis and design will be used to achieve these improvements and to improve other aspects of the CASES software. The software was also enhanced to improve the software's testability (operability, observability, controllability, decomposability, and simplicity).

Finally, CASES 1.1 has a number of shortcomings and requires a number of improvements in order to support all of the ideas and concepts in the HFSE [PUET03]. CASES dependency relationships need to be extended to incorporate the Software Quality Deployment (SQFD) methodology to provide a richer dependency spectrum (type and degree) between software development steps and components.

D. QUALITY FUNCTION DEPLOYMENT (QFD)

1. Basic QFD [CLAU88]

a. Summary

Clausing, one of the pioneers of QFD usage in the United States, points out that QFD's primary goal is to overcome three major problems: 1) disregard for the "voice of the customer", 2) loss of information, and 3) different individuals and functions working to different requirements. QFD consists of well-developed formats (matrices and charts) and a style of organizational behavior that facilitates a holistic response to customer needs [CLAU88]. QFD can benefit an organization by increasing the company's market share and profit. It does this through reduced development and manufacturing costs, improved product quality, provision of features that satisfy customer need, and reduced development time. QFD has proven very successful in producing products that appeal to customers. Metaphorically, the customer speaks one language and the manufacturer speaks another. QFD provides linguistic continuity from customer to manufacturer and brings corporate knowledge to bear on the product that achieves multifunctional consensus.

b. Key Idea

The key idea presented in [CLAU88] is that QFD is needed to deploy customer needs throughout the corporate communication circle and return to the customer a new product that fully meets those needs. QFD is a product development methodology that systematically deploys customer requirement priorities into the product design and guides production operation on the factory floor. QFD provides a win-win development approach for the manufacturer and customer.

c. Concepts Useful in the Thesis

Although these ideas about QFD have been developed for building quality products in the product industry, there are compelling reasons for integrating this methodology in software development processes. This key concept provides a foundation for this thesis research. It provides the necessary information to transfer these product manufacturing management techniques into a holistic model for software acquisition and development.

The QFD benefits provide partial solutions to the following problems:

(1) The Software Crisis. This crisis is caused by the difficulty in and the failure of successfully evolving complex software systems.

(2) Increased Demand for Quality Software. Software demand is increasing at an astounding rate; industry has found it difficult to find the necessary professional talent required for meeting this software development demand.

(3) Inadequate Customer-Developer Communication. Better communication between software customers and developers is needed. There is currently inadequate communication of requirements and risk throughout the development life cycle.

(4) Lack of High Assurance Systems. Industry requires ever increasing numbers of high-assurance software systems, particularly in mission-critical activities, business financial transactions, and life-critical medical applications.

2. House of Quality (HOQ) [HAUS88]

a. Summary

In [HAUS88], Hauser and Clausing present the “house of quality” (HOQ) as the basic implementation construct of the management approach known as Quality Function Deployment (QFD). The “house of quality” (see Figure 13) provides a conceptual, abstract view of a product design and provides the means for inter-functional planning and communication. The authors point out that the main challenge in design (product design, software design, etc.) is to learn from customer experience and reconcile what customers say they want with what engineers can reasonably build. The house of quality provides such a mechanism for product design, development, and manufacture. Traditionally, the house of quality has been used in the automobile industry and other factory environments, but the same challenge of managing design complexity that QFD tackles in the product industry also plagues the software development industry.

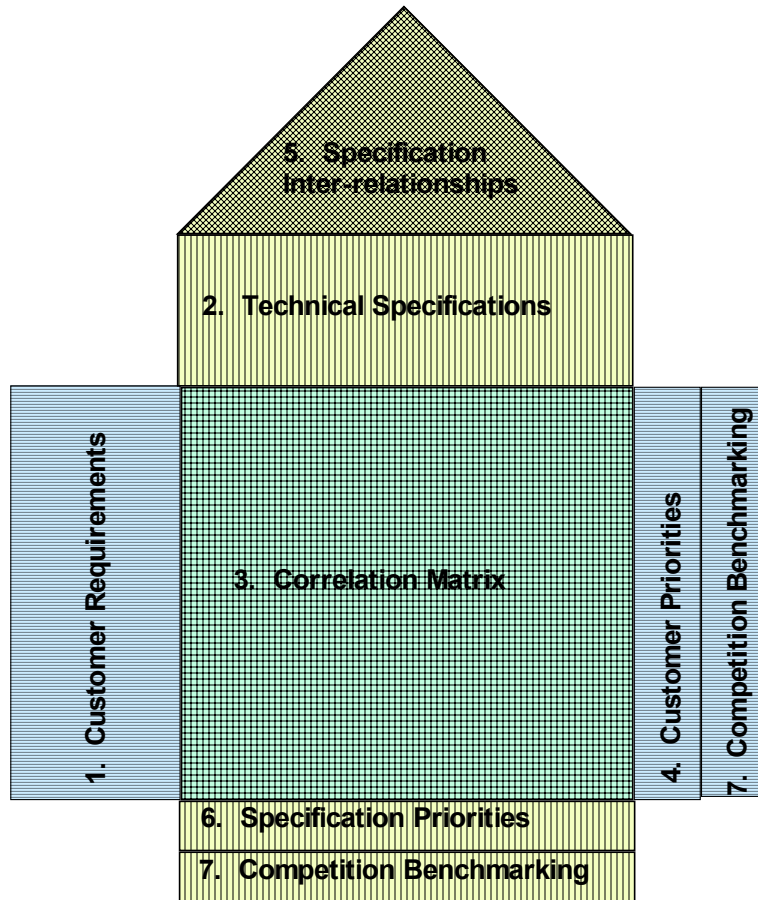


Figure 13 First Level QFD Matrix -- The "House of Quality" (after [PUET03])

Building the initial house of quality consists of identifying and capturing customer attributes (CAs), their relative importance, customer perceptions, engineering characteristics (ECs), relationships between attributes, objective measures, and specification trade-offs. Customer attributes are the customer requirements, which describe products and product characteristics. The relative importance provides weighting of the customer attributes. Weightings are based on design team members' direct experience with stakeholders, statistical techniques, observations in the customer's workplace, and/or surveys. Customer perceptions are used to identify where a new product could match or exceed a competitor's product (also known as competition benchmarking). Benchmarking helps to illustrate where a planned product stands relative to potential competition. The ECs describe the product in the language of the engineer.

ECs should describe the product in measurable terms that directly affect customer perceptions. The relationship matrix indicates how much each EC affects each CA. The indications can be based on expert engineering experience, customer response, and statistical data. The objective measures are values, which describe the link between a particular EC and a CA for the product and some of its competitors. The roof matrix assists engineers with developing a creative solution and balancing of objectives by identifying specific engineering trade-offs.

The house of quality is further improved by adding technical difficulty, inferred (or calculated) importance, estimated cost, and targets for the objective measures. The “deployment” portion of the QFD methodology requires the engineer to create additional QFD matrices between subsequent artifacts in the development process. Additionally, the houses can be linked to convey the customer’s voice through to manufacturing. As stated by [PUET03]:

While this first QFD matrix is easily constructed, the real strength of the QFD methodology occurs after the completion of the highest-level matrix. As the project continues, additional matrices are established, each of which establishes dependencies with the original stakeholder requirement priorities. This provides visibility of what is important and what is not.

This linkage provides many levels of abstraction, which are critical for software development. As the house of quality is improved with additional technical difficulty and growth, it becomes more complex due to the magnitude of CAs and ECs and calculations required. [HAUS88] states:

An elegant idea ultimately decays into process, and processes will be confounding as long as human beings are involved. But that is no excuse to hold back. If a technique like the house of quality can help break down functional barriers and encourage teamwork, serious efforts to implement it will be many times rewarded.

b. Key Idea

The key idea in [HAUS88] is that “the house of quality” provides a basic implementation tool for managing QFD throughout the design, development, and

manufacturing of a product. Underlying the "house" however, are fundamental design entities and relationships currently captured in matrices, which can be embodied in other constructs (such as a directed hypergraph). The QFD house of quality is worthy of implementation as software development support tool because it is a methodology which adequately communicates customer requirements throughout a software engineering design. This linkage provides many levels of abstraction, which are critical to understanding complex software systems. QFD accomplishes this through the series of HOQ matrices where each matrix becomes increasingly specific. This linkage provides a holistic framework, which can be used to guide their dependency choices. Additionally, this linkage helps developers reduce software development time, increase customer satisfaction, lower software cost, create greater synergy between components (e.g. requirements, specification, code modules, etc.), establish better software development memory, and introduce fewer software defects into the process. The initial HOQ starts with the "voice of the customer" stated in plain English. From this matrix, requirements, specifications (possibly using formal specification language), and eventually code modules (as well as other artifacts) are derived. Additionally, as [CLAUS88] states:

The longer the development cycle, the more likely it is that the marketplace and associated customer needs will change. The change will most likely be a quantitative change – that is, important levels or satisfaction performance levels on the customer needs may vary. Less likely, but very significant if it happens, is a qualitative change: a new customer attribute may appear. The QFD matrix or matrices can easily be updated with such changes, and development targets and priorities can be reassessed to determine whether the development work as planned is still on track.

The question is, however, whether the "house" is the appropriate and most efficient construct for capturing and communicating these requirements in a software engineering setting or whether another construct would be more effective.

c. Concepts Useful in the Thesis

The key concepts in [HAUS88] useful to this thesis are the "house" construct and the linkages of "houses" throughout a development process. These concepts provide a foundation for this thesis research as well as suggest an architecture to implement QFD within CASES. While the QFD construct may not be the most efficient

approach for capturing and communicating artifacts of software evolution, CASES already contains the relational hypergraph as an evolution construct, which can be modified to capture the similar information used in the HOQ. The software architecture is as critical as the software product itself. The house gives this thesis an established foundation to integrate QFD and CASES. If a customer has a need for unique domain-specific software, the house has many flexible characteristics, which can be modified to accommodate the uniqueness of the development effort resulting in a quality software development effort.

3. Software Quality Function Deployment [ZULT92]

a. Summary

In [ZULT92], Zultner discusses the importance of Quality Function Deployment (QFD) for software. He introduces QFD and how it can improve both product and process with impressive results. Zultner introduces the concept of “customer” deployment. The Customer Deployment component of QFD is an important attribute for software, because one software product might need to satisfy many different types of customers. A special challenge exists for complex products – those involving a combination of hardware, software, and service (not to mention being a subsystem of a complex system). Tools help to deal with this complexity: matrix data analysis charts (or priority matrices), affinity diagrams, relations diagrams, hierarchy diagrams, matrices, precedence diagrams, and process decision program diagrams. These tools can be used to improve the process, product, and vision. These improvements include improvements in quality, technology, cost reduction, reliability, and development work ethic.

Coherent software development is an important byproduct of QFD. Traditional development is incoherent. QFD focuses the development effort on those aspects of the design that are of greatest importance to the customer(s). QFD maintains this focus throughout the entire development process, from requirements to design, to coding, to documentation. Kano et. al. (as cited in [ZULT92]) provides the Kano model that characterizes three types of requirements (exciting, normal, and expected) based on their influence on customer satisfaction (see Figure 14). QFD is critical because customers cannot typically articulate all of their exciting and/or expected requirements.

QFD provides a customer centered software development approach that helps in this requirements elicitation process. Customer requirements can be explored through specification (why customer wants it), exploration (what is required to build great software), and design (how to build great software).

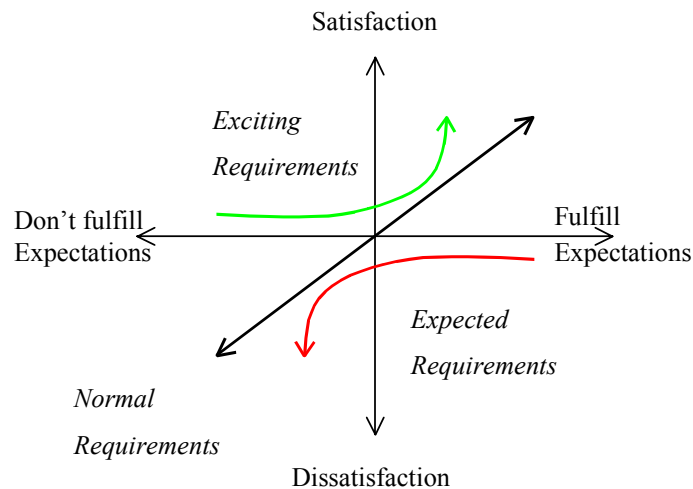


Figure 14 Kano Model (after [ZULT92])

QFD provides a process that conveys requirements throughout a development effort's data and process models. This process includes four steps: capturing raw customer expressions; translation of those expressions into clear, concise, and concrete objective items; organization of the expressions and objective items into a hierarchy that is meaningful to the customer; and prioritization of requirements (e.g. 1 to 5 scale, or by a more advanced method). Customer requirements can come from many sources and consist of several types (exciting, normal, and expected). Software engineers must meet these requirements to build excellent software. Expected requirements deliver on expectations that the customer will not be disappointed. Normal requirements are the easiest customer requirements to uncover and generally have a direct relationship to customer satisfaction – giving the customer more of a “normal” requirement satisfies him more, giving him less satisfies him less. Exciting requirements are the most difficult requirements to uncover, but “wow” the customer instead of just satisfying them. Kano's model of how these three types of requirements correspond to customer satisfaction is shown in Figure 14.

In order to deploy the “voice of the customer” throughout the software development process it is necessary to employ a chain of matrices (see Figure 15). In this case, the analysis and design is shown as a five phased process of matrices consisting of: customer requirements/specifications, specifications/software modules, software modules/code, code/unit tests, and unit test/customer requirements matrices (see Figure 15).

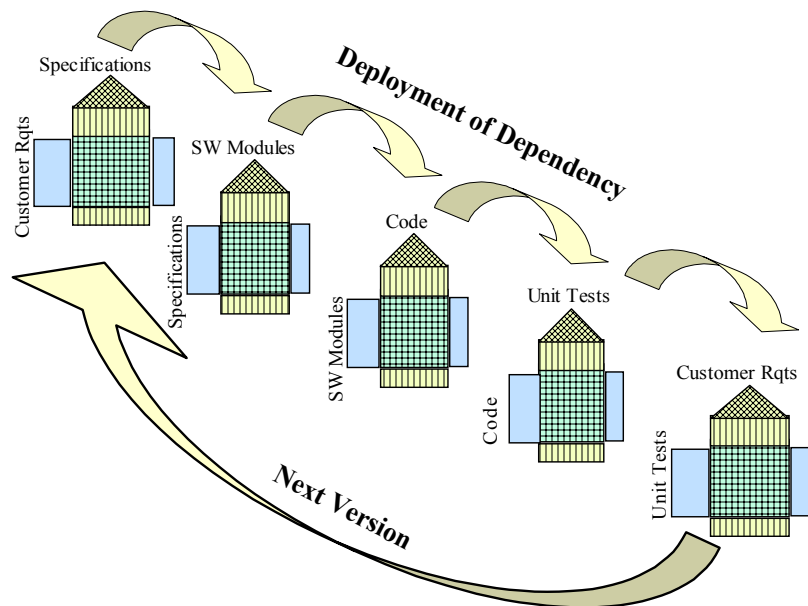


Figure 15 Example of a QFD Matrix Deployment (from [PUET03])

Figure 15 illustrates a simplistic software development life cycle, in which the chain of matrices provides the “deployment” of expected, normal, and exciting requirements throughout the software development process.

b. Key Ideas

The key idea in [ZULT92] is that QFD is a development process to improve software products, process, and strategy. This is accomplished by making software development more coherent, building quality into the product, and providing

rationale for development decisions. Tools are required to deal with the complexity of developing software systems with QFD.

QFD makes software development more coherent so that the best effort of one phase of the software development process feeds the best effort of the next phase of the process, and so on. Each of these best efforts can be directly traced back to what the customer views as the most important part of the design and forward to future components of the design. QFD gives the software development effort a solid foundation for embedding quality into the product.

Software quality can take on many different forms (e.g. functionality, efficiency, reliability, usability, maintainability, and portability), but [PRES01] emphasizes the following three important points about software quality:

- Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.
- Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
- A set of implicit requirements often goes unmentioned (e.g., the desire for ease of use and good maintainability). If software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspect.

QFD provides a methodology for handling these important points. QFD identifies and implements positive values of customer satisfaction (based on Kano's model). [PUET03] emphasizes "Traditional software engineering has generally focused on just removing the "dis-satisfiers" i.e. the defects – this approach is necessary, but not sufficient!" QFD is critical to software development because it provides a methodology for handling the important characteristics of software quality.

QFD provides the mechanism for the deployment of quality throughout a software design through the use of linked houses of quality. This linkage helps

developers reduce software development time, increase customer satisfaction, lower software costs, create more efficient cooperation between components, establish coherent software development evolution history, and make fewer mistakes. Additionally, this linkage can help to uncover implicit requirements (not stated directly by the customer) through the correlation of requirements and specifications.

c. Concepts Useful in the Thesis

QFD is ideal for software evolution because it focuses on improving the process, product, and strategy throughout a product's life cycle. Prioritization is a step in QFD for conveying requirements through the software development process. The use of tools to reduce complexity of this key concept is essential. CASES 1.1 is an automated tool that can be modified to incorporate QFD. The embedding of QFD into CASES results in the prioritization based on weighted steps within the Relational-Hypergraph model. A pair of components and a step can be represented by a QFD matrix. Matrices between all component tuples in the development process can be chained to ensure the propagation of a dependency.

Additionally, this chain of matrices can be modeled as a software development process diagram. CASES refers to such diagrams as *Project Schemas*. Using the software development process outlined in Figure 15, the project schema in Figure 16 is a model of the chain of matrices.

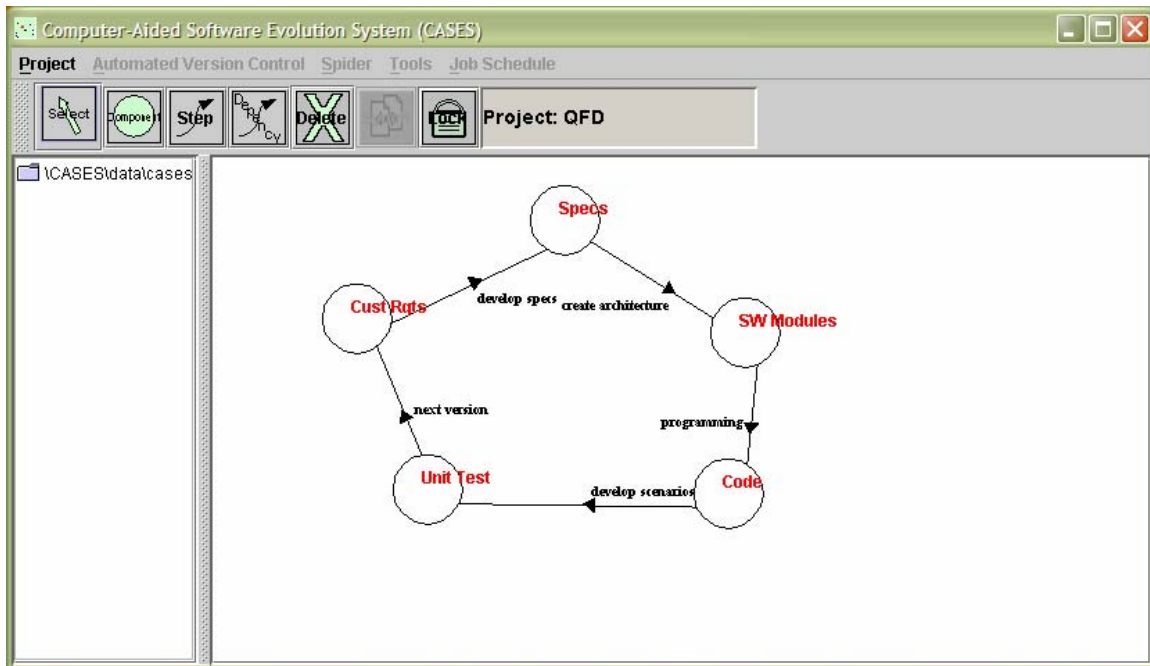


Figure 16 QFD Model drawn as a Process Diagram

In Figure 16, the circles (e.g. Cust Rqts, Specs, etc.) represent the artifacts created in the software development process and the edges (e.g. develop specs, create architecture, etc.) represent the software development activities that create new artifacts. A project schema models the software evolution process and manages the complex QFD matrices generated throughout development.

E. RATIONAL PROJECTCONSOLE [RATI01]

1. Summary

Rational ProjectConsole is an application that is related to the work in this thesis in that it attempts to accomplish some of the same aims. This application provides views of information through a web-based solution versus a Java-based application such as CASES. In some ways, it is an attempt to provide a holistic framework for the project management of software evolution and development when using Rational Software Corporation's software development suite of tools and some other specific third party tools. As stated by [RATI01]:

ProjectConsole is a server-based application that publishes artifacts from Rational Suite and third-party tools to a project Web site. The ProjectConsole server knows where the artifacts live and how to collect, format, and present the content using customizable templates. ProjectConsole is tightly integrated with Rational Suite development tools so that it can hyperlink all project artifacts as it automatically creates and updates the project Web site.

2. Key Ideas

Rational ProjectConsole is an attempt to provide similar information as that provided by the Holistic Framework for Software Evolution. The server-based application is capable of publishing artifacts from software development tools and contains progress metrics for the development of project artifacts. The artifacts can be mapped to graphical representations to illustrate the state of the project through the ProjectConsole Dashboard (see Figure 17). The Dashboard can illustrate empirical data related to priority, trends, defects, etc. and allows the user to drill down to locate the specifics of the artifact's information. Unfortunately, if the user requires knowledge about which artifacts correspond to the customer's highest priorities (or any other specific dependency) then the lack of a "true" holistic framework becomes apparent. ProjectConsole falls short of providing insight into dependency relationships between software artifacts outside of the source of the artifacts metrics (e.g. the number of high priority requirements) or a temporal dependency (e.g. trends).

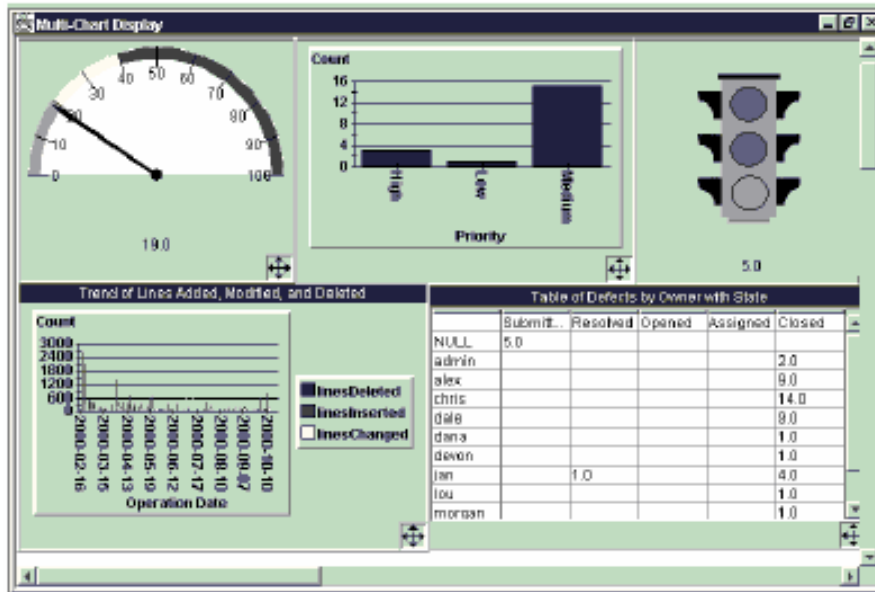


Figure 17 ProjectConsole Dashboard (from [RAT101])

3. Concepts Useful in the Thesis

Progress is being made to build applications to assist with the holistic framework demands, but still fall short of a “true” holistic framework for software engineering. This thesis takes the concepts of retrieving, viewing and analyzing project artifacts to a holistic framework level. CASES can trace dependencies between artifacts through types, version, variance, and components.

F. CHAPTER SUMMARY

This chapter presents a summary, key ideas, and useful concepts from a scoped literature review of foundation work in the research area. This thesis builds on these main research areas and researchers:

- Holistic Framework for Software Evolution [PUET03]
- Relational Hypergraph Software Evolution Model [HARN99a]
- CASES 1.1 [LEHC99]
- Quality Function Deployment [CLAU88], [HAUS88], [COHE95] and [ZULT90, 92, 93]

Additionally, this chapter presented an overview and differentiation of the related work of Rational ProjectConsole [RATI01], which provided tangible evidence of a need for the Holistic Framework for Software Evolution.

THIS PAGE INTENTIONALLY LEFT BLANK

III. QFD DEPENDENCY RELATIONSHIPS

A. THE HOUSE OF QUALITY (HOQ)

1. Introduction

CASES provides the stakeholder with QFD dependency matrices (houses) based upon the software evolution model created within the GUI draw pane and dependencies created using the dependency creation dialog. The HOQ is a simple but powerful tool that is positioned at the heart of QFD. CASES employs JTables to create the HOQ for computing calculations. The HOQ consist of three parts: 1) the “what” table (left-most), 2) the “how” table (top-most), and 3) the correlation table (middle). The HOQ calculation feature is a vital tool for propagating dependency values throughout the software evolution model life cycle.

2. Creating Models

After creating a project using the “project” menu, a stakeholder will need to model a software evolution development process (see Figure 18) within the CASES draw pane. This model is known as the *Project Schema* and is a high level abstraction of the evolutionary software development process that the engineer is employing. Also, as discussed in Chapter II, this schema is an abstraction of the linked series of matrices that form the underlying QFD “deployment” of the development effort. The stakeholder creates the project schema by drawing components and connecting them with steps. Once two or more components are created, steps can be created between them. An arrow will appear to depict the direction of that step’s activity (i.e. which components lead to the creation of new components via the step). The user can customize steps and components by editing the names and descriptions within the respective creation dialogs. After creating a model, the model must be locked before versions and variants can be constructed.

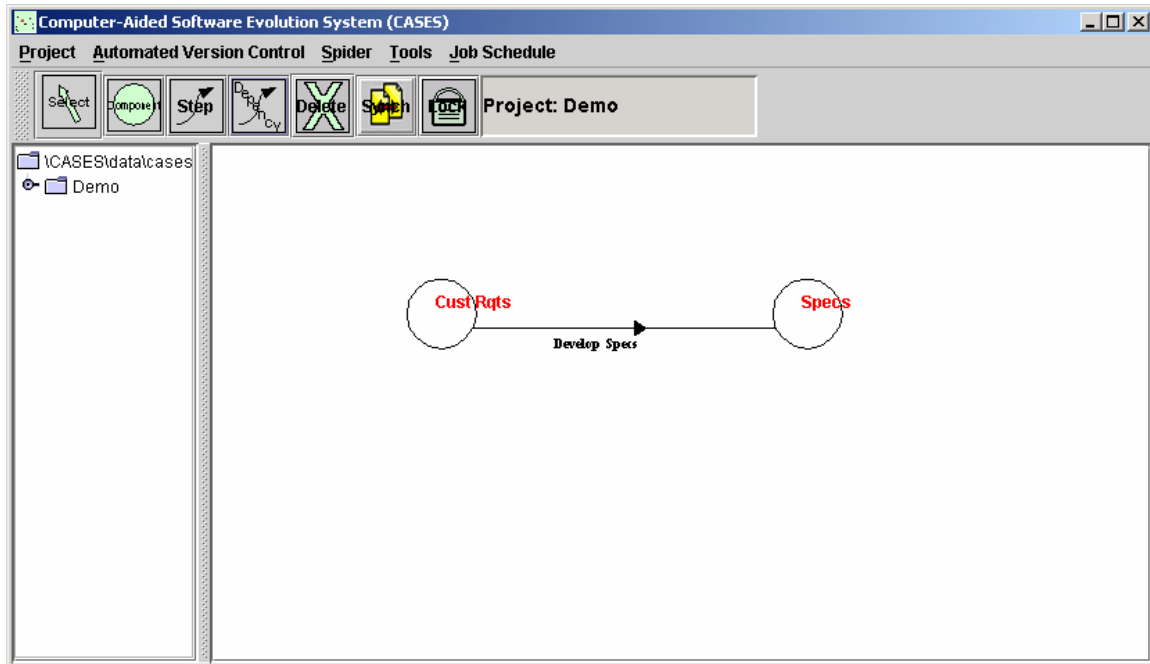


Figure 18 Project Schema Creation

3. Creating Dependencies

After creating a project schema, a stakeholder will need to create dependencies that are relevant to the software evolution model. The user inputs the dependency name, description, type, value range, default value and origin in the dependency dialog (see Figure 19). The name provides a short name for identifying one dependency from another. The description is available to elaborate on the particulars of a dependency. The type can be selected from “risk”, “safety”, and “parent-child”. The “type” attribute currently does not provide any functionality in this version, but is provided for future extensions in which specific, pre-defined dependency type attributes can be associated at run-time with particular instantiated dependencies. Additionally, the value range is provided for future extensions and currently assumes a real value of 0 to 9. Future extensions of value ranges could account for other ranges of real, integer, or Boolean values. The default value is used to initialize the dependency values in the “what” and “how” JTables upon the first use of a dependency. The *origin* must be set at a specific component and represents the basis from which the dependency values are generated. For instance, a dependency of “Requirement Risk” might use the “Requirement”

component as the origin. This origin is used to perform upstream and downstream calculations (discussed later) originating from that component.

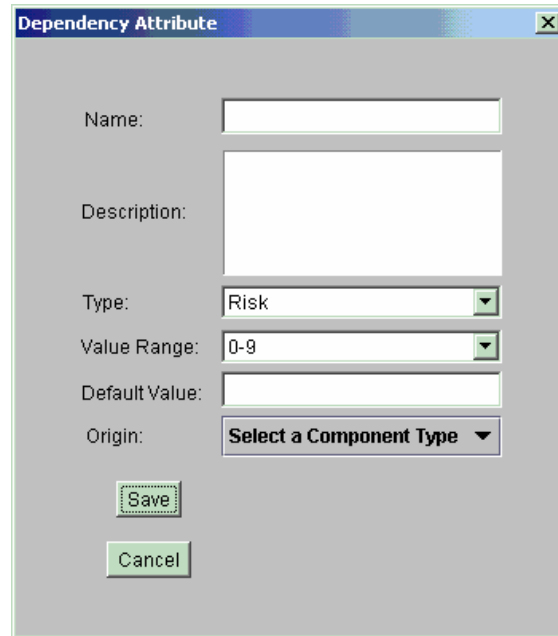
The image shows a software dialog box titled "Dependency Attribute". It has a standard Windows-style title bar with a close button. The dialog contains several input fields: "Name:" with a single-line text box; "Description:" with a multi-line text area; "Type:" with a dropdown menu currently showing "Risk"; "Value Range:" with a dropdown menu currently showing "0-9"; "Default Value:" with a single-line text box; and "Origin:" with a dropdown menu currently showing "Select a Component Type". At the bottom of the dialog are two buttons: "Save" and "Cancel".

Figure 19 Dependency Dialog

Similarly, a dependency of “Specifications Difficulty” might rely on the Specifications as the origin. The dependency origin fixes the logical component upon which a particular dependency is initially defined. Dependency values are then “deployed” forward (downstream) and backward (upstream) through the development effort from this origin.

B. CALCULATIONS

1. Deployment Equations

The propagation of dependencies is accomplished through a series of matrix multiplications that propagate from the origin component upstream and downstream through steps to all connected components. Downstream calculations are performed with the flow of the step (in the general temporal direction of the software development process) while upstream calculations are performed against the flow of the step (temporally backwards in the development process). For example, if a project schema consisted of requirements components **R** leading to the generation of specification

components S , via a step called *specification generation*, then *downstream* would be considered as R to S , while *upstream* would be considered S to R . The mathematics of dependency deployment in [PUET03] provide the underlining theory behind CASES' implementation. The calculations can be preformed individually or collectively. The stakeholder can perform individual calculations (upstream or downstream) any time through the use of the "Calc" button on the HOQ window (see Figure 20). This will deploy the dependencies values within the single open QFD dialog. The user can perform collective calculations through the use of the "Sync" button on the CASES toolbar (see Figure 18). The "Sync" button will deploy dependency values upstream and downstream from the origin's version throughout all QFD matrices (even those not open). For example, version 1.1 will deploy all dependency values downstream to other existing versions -- version 1.1 is the initial version created by CASES. The transition between versions is handled through *bridging steps*. A bridging step is the last component connected to the first component of the project schema, where the last component is of version $x.y$ and the first component is of version $pred(x.y)$ or $succ(x.y)$. Note that $pred(value)$ is a function that returns the previous version of the value parameter and $succ(value)$ is a function that returns the next version of the value parameter (if such a version exists).

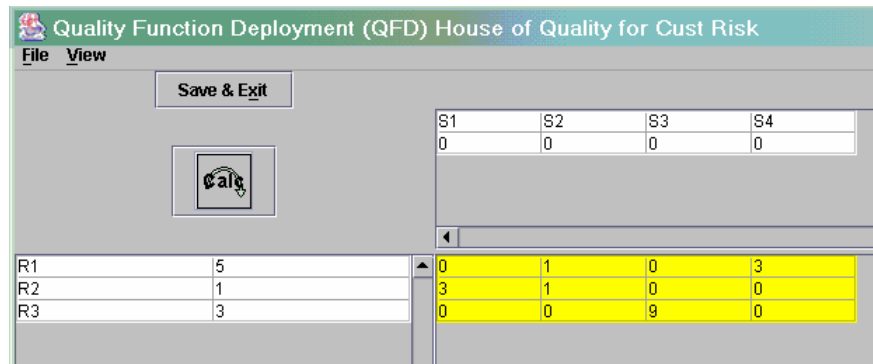


Figure 20 QFD Matrix for Risk Deployment Example

The general algorithms for a downstream and upstream calculation are provided below with an example of each. These algorithms will be explained using requirements and specifications as the example software evolution components with a risk dependency.

2. Downstream Calculations

Let $\mathbf{R}$ be a vector of values of a risk dependency of order m where each value of $\mathbf{R}$ corresponds to a particular requirements component.

$$\mathbf{R} = [r_1, r_2, \dots, r_m] \quad \text{Equation 1}$$

Let *weight* w be a scalar equal to the sum of the requirement risk dependency values:

$$w = \sum_{i=1}^m r_i \quad \text{Equation 2}$$

Let $\mathbf{C}$ be a correlation matrix of order $m \times n$ between the requirements (of order m) and specifications (of order n):

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} & & \\ c_{21} & c_{22} & & \\ & & \ddots & \\ & & & c_{mn} \end{bmatrix} \quad \text{Equation 3}$$

Let $\mathbf{S}'$ be the specification vector result from the cross product of vector $\mathbf{R}$ and matrix $\mathbf{C}$ with order n :

$$\begin{aligned} \mathbf{S}' &= \mathbf{RC} \quad \text{or} \quad \mathbf{S}' = [s'_1, s'_2, \dots, s'_n] \\ \text{and} \quad s'_j &= \sum_{i=1}^m r_i c_{ij} \quad \text{for } j = 1 \dots n \end{aligned} \quad \text{Equation 4}$$

Let *total* t be the scalar sum of the values of $\mathbf{S}'$:

$$t = \sum_{j=1}^n s'_j \quad \text{Equation 5}$$

Therefore, using the above equations the downstream calculation finds $\mathbf{S}$ which equals the normalized specification vector of $\mathbf{S}'$:

$$\mathbf{S} = [s_1, s_2, \dots, s_n] \quad \text{where } s_j = s'_j * \frac{w}{t} \quad \text{Equation 6}$$

Within CASES 2.0 downstream calculations are implemented by the following code:

```
for (int i = 0; i < col; i++) {
    sum[i] = 0.0;
    for (int j = 0; j < row; j++) {
        double amount = 0.0;
        try {
            amount = gdv.getValue(depMatrix.getValueAt(j, i));
        }
        catch (java.lang.NumberFormatException e2) {
            System.out.println("Value error in dependency Matrix");
        }
        sum[i] += amount * gdv.getValue(leftMatrix.getValueAt(j, 2));
    }
    total += sum[i];
}
// normalize sums
for (int i = 0; i < col; i++) {
    sum[i] /= total; // get ratio to other sums
    sum[i] *= weight; // normalize to the weight of the left elements
    double temp = (gdv.roundUp(sum[i])) / this.vectorSize; // round result up and format for output
    topMatrix.setValueAt(new Double(temp), 2, i);
}
```

Figure 21 Downstream Dependency Calculations Code

CASES expects a double value within the dependency matrices but will automatically convert integers input values into double values. CASES will throw the “number format exception” if the value is not of the appropriate format.

3. Downstream Calculations Example

The following example using CASES is provided as a downstream dependency deployment example and parallels a similar example in [PUET03]. A “Cust Risk” dependency was created with the origin set at the “Cust Rqts” component (see Figure 18). Figure 20 shows a “what” JTable of three requirements {R1, R2, R3} with associated risk values of {5, 1, 3}; a “how” JTable of four specifications {S1, S2, S3, S4} without any associated risk values; and a correlation JTable of {{0, 1, 0, 3}, {3, 1, 0, 0}, {0, 0, 9, 0}}.

Figure 22 illustrates the downstream calculation results after the user presses the “Calc” button. The four specifications now have associated risk values of {0.53, 1.06, 4.76, 2.65}. These values provide a clear view of how requirements risk is deployed to specifications. Requirements propagated their risk to the specifications, with the biggest

influence going to S3 and S4. Intuitively this might be obvious for this small example, but when there is a web of numerous artifacts (e.g. requirements, specifications, etc.) the power of automated support for QFD dependency deployment becomes paramount. Additionally, the ability to propagate the specification risk values further downstream (not to mention the upstream capabilities) makes this tool extremely valuable.

S1	S2	S3	S4
0.53	1.06	4.76	2.65

R1	R2	R3
5	1	3

0	1	0	3
3	1	0	0
0	0	9	0

Figure 22 QFD Matrix for Requirements Risk Deployment Downstream Calculation

4. Upstream Calculations

Let $\mathbf{S}$ be a specification vector of values of a risk dependency of order n :

$$\mathbf{S} = [s_1, s_2, \dots, s_n] \quad \text{Equation 7}$$

Let *weight* w be a scalar equal to the sum of the requirement risk dependency values:

$$w = \sum_{j=1}^n s_j \quad \text{Equation 8}$$

Let $\mathbf{C}$ be a correlation matrix of order $m \times n$ between the requirements (of order m) and specification (of order n):

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} & & \\ c_{21} & c_{22} & & \\ & & \ddots & \\ & & & c_{mn} \end{bmatrix} \quad \text{Equation 9}$$

Let $\mathbf{R}'$ be the requirements vector result from the cross product of vector $\mathbf{R}$ and matrix $\mathbf{C}^T$ with order m :

$$\mathbf{R}' = \mathbf{S}\mathbf{C}^T \quad \text{or} \quad \mathbf{R}' = [r'_1, r'_2, \dots, r'_m] \quad \text{Equation 10}$$

Let *total t* be the scalar sum of the values of $\mathbf{R}'$:

$$t = \sum_{i=1}^m r'_i \quad \text{Equation 11}$$

Therefore, using the above equations the upstream calculation finds $\mathbf{R}$, which equals the normalized specification vector of $\mathbf{R}'$:

$$\mathbf{R} = [r_1, r_2, \dots, r_m] \quad \text{where} \quad r_i = r'_i * \frac{w}{t} \quad \text{Equation 12}$$

Upstream calculations are implemented by the code in Figure 23.

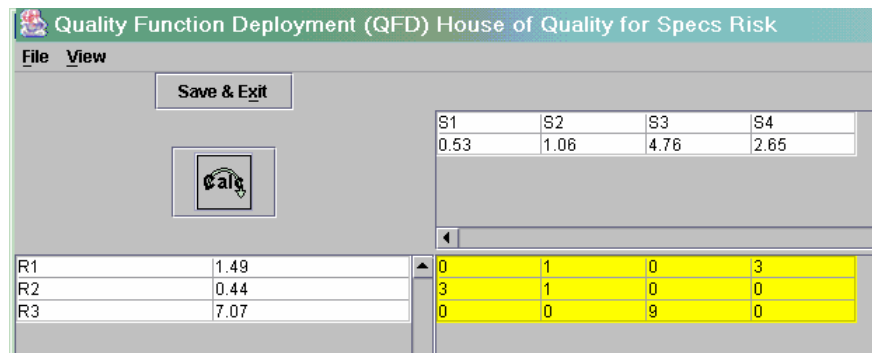
```
for (int i = 0; i < row; i++) {
    sum[i] = 0.0;
    for (int j = 0; j < col; j++) {
        double amount = 0.0;
        try {
            amount = gdv.getValue(depMatrix.getValueAt(i, j));
        }
        catch (java.lang.NumberFormatException e2) {
            System.out.println("Value error in dependency Matrix");
        }
        sum[i] += amount * gdv.getValue(topMatrix.getValueAt(2, j));
    }
    total += sum[i];
}
// normalize sums
for (int i = 0; i < row; i++) {
    sum[i] /= total; // get ratio to other sums
    sum[i] *= weight; // normalize to the weight of the left elements
    double temp = (gdv.roundUp(sum[i])) / this.vectorSize; // round result up and format for output
    leftMatrix.setValueAt(new Double(temp), i, 2);
}
```

Figure 23 Upstream Dependency Calculation Code

Again, CASES expects a double value within the dependency matrices and will automatically convert integer input values to double values. CASES will throw the “number format exception” if the value fails to meet the appropriate format.

5. Upstream Calculations Example

The following example using CASES provides an upstream dependency deployment example and parallels the example presented in [PUET03]. A “Specs Risk” dependency was created with the origin set at the “Specs” component (see Figure 18). Figure 24 illustrates the upstream calculation results after the stakeholder presses the “Calc” button. The three requirements now have associated risk values of {1.49, 0.44, 7.07}. Note that these values are not the same as the starting values for the downstream calculation example {5, 1, 3}, because these values preserve the view perspective from the specifications components. In other words, R3 and R1 respectfully influenced the specification the most. This is an important feature when a stakeholder is attempting to improve the software engineering processes (as expected for a SEI CMM level 3 or above organization).



	S1	S2	S3	S4
	0.53	1.06	4.76	2.65

	S1	S2	S3	S4
R1	0	1	0	3
R2	3	1	0	0
R3	0	0	9	0

Figure 24 QFD Matrix for Specification Risk Deployment Upstream Calculation

C. THE ROOF

1. General

The “roof” of the House of Quality establishes the dependency of a particular type of atomic components between each other. The Dependency Network Diagram (see Figure 25) illustrates the internal relationships between all of the requirement atomic components. R2 has a strong negative dependency on R1; R2.1 has a weak positive dependency on R2, R2.2 has a strong positive dependency on R2 and R2.1; R2.3 has a moderate positive dependency on R2.2, a moderate negative dependency on R1, and a

weak positive dependency on R2; and R3 is independent. This construct is useful to identify the *drivers* and *indicators* among atomic components.

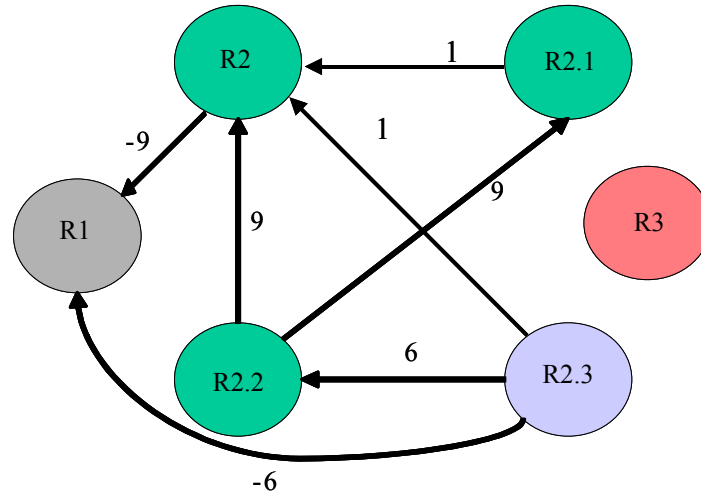


Figure 25 Dependency Network Diagram

Notice that R2.3 in Figure 25 has outgoing arrows and no incoming ones. This is an indication that R2.3 is a *driver* in the sense that it influences other atomic components but is not in turn dependent upon any other atomic components. On the other hand, R1 in Figure 25 has only incoming arrows and no outgoing ones. Therefore, R1 is an *indicator* in the sense that it does not influences other atomic components but is in turn dependent upon other atomic components. Note that requirement R3 has no links to any other requirements. This means that it is neither an indicator nor driver.

2. Roof Example

A dependency network diagram (see Figure 25) can be represented in CASES using the JTable construct (see Figure 26). Drivers and indicators can be stored in CASES 2.0, but this version does not exploit that part of the House of Quality for reducing QFD matrices (e.g. indicators may be removed from matrices because they are dependent on the drivers which are represented). If both the row and column values of the roof matrix for a given atomic component are entirely zero then that atomic component is neither a driver nor an indicator (e.g. R3 in Figure 26). If a column (and not its row) is all zeros then the atomic component is an indicator (e.g. R1 in Figure 26).

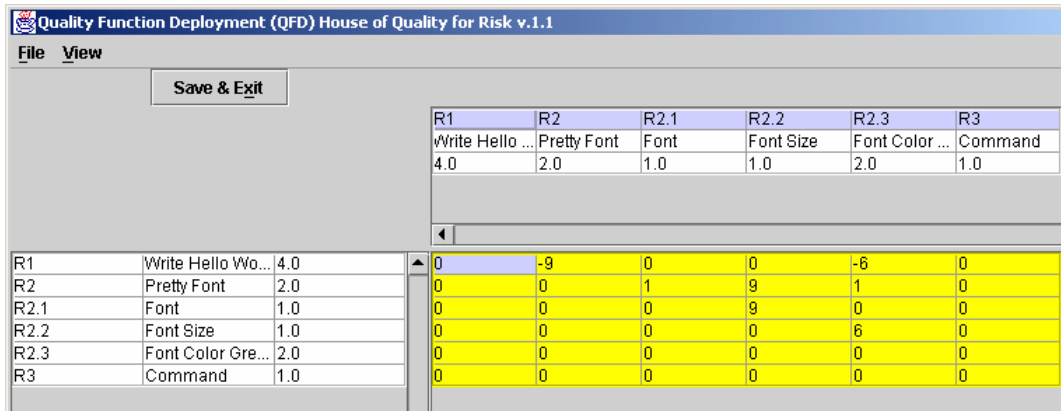


Figure 26 HOQ Roof

If a row (and not its column) is all zeros then the atomic component is a driver (e.g. R2.3 in Figure 26). The diagonal of the correlation matrix must remain zero due to the fact that components are not dependent upon themselves. The dependency cell direction for a 6x6 matrix is listed in Table 3 (where “->” indicates the direction of the dependency). Each dependency cell’s direction is from top atomic component to left atomic component. This technique allows the stakeholder to capture the direction, influence (positive or negative), and magnitude of a dependency within a single cell without using any special characters (e.g.”->”) or graphics.

	R1	R2	R2.1	R2.2	R2.3	R3
R1	N/A	R2->R1	R2.1->R1	R2.2->R1	R2.3->R1	R3->R1
R2	R1->R2	N/A	R2.1->R2	R2.2->R2	R2.3->R2	R3->R2
R2.1	R1->R2.1	R2->R2.1	N/A	R2.2->R2.1	R2.3->R2.1	R3->R2.1
R2.2	R1->R2.2	R2->R2.2	R2.1->R2.2	N/A	R2.3->R2.2	R3->R2.2
R2.3	R1->R2.3	R2->R2.3	R2.1->R2.3	R2.2->R2.3	N/A	R3->R2.3
R3	R1->R3	R2->R3	R2.1->R3	R2.2->R3	R2.3->R3	N/A

Table 3 Roof Dependency Direction

D. USER-DEFINED VIEWS

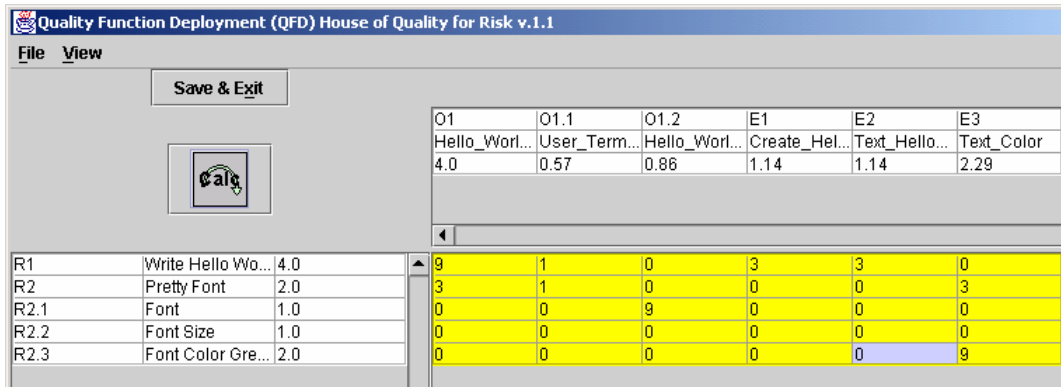
1. Dependency Threshold

Components within a software evolution process could consist of several hundreds of atomic artifacts. Therefore, to make QFD within CASES a pragmatic tool for decision support – the Dependency Threshold view was created to provide a scoped

view of dependency relations. This capability allows the user to conduct dependency *trimming* (see [PUET03]) to trim the view of a full QFD matrix.

2. Dependency Threshold Example

Given the matrix in Figure 27, the stakeholder can perform a Dependency Threshold operation to produce the view based upon a statistical equation related to the mean and standard deviations of the dependency vector (see Equation 13).



	O1	O1.1	O1.2	E1	E2	E3
Hello_Worl...	User_Term...	Hello_Worl...	Create_Hel...	Text_Hello...	Text_Color	
4.0	0.57	0.86	1.14	1.14	2.29	

	R1	R2	R2.1	R2.2	R2.3
Write Hello Wo...	4.0				
Pretty Font	2.0				
Font	1.0				
Font Size	1.0				
Font Color Gre...	2.0				

9	1	0	3	3	0
3	1	0	0	0	3
0	0	9	0	0	0
0	0	0	0	0	0
0	0	0	0	0	9

Figure 27 Dependency Threshold Example

Figure 28 allows the user to trim the view based upon Equation 13 where t is the dependency threshold value specified against the mean and standard deviation of each group of components, x is a user specified scalar, and i represents each different set of components:

$$t = \mu_i \pm x\sigma_i \quad \text{Equation 13}$$

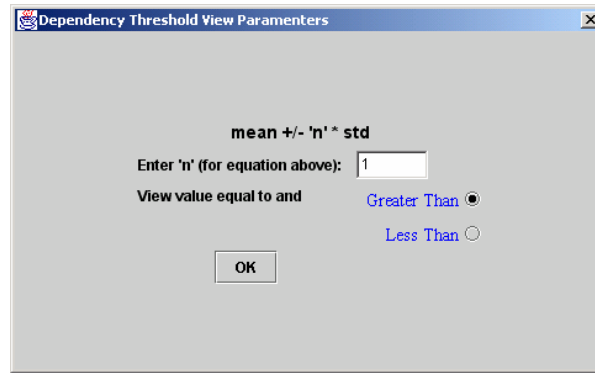


Figure 28 Dependency Threshold Dialog Window

Figure 29 illustrates the *dependency threshold view* where R1 and O1 are shown in the resultant matrix, which met the constraints of the threshold equation.

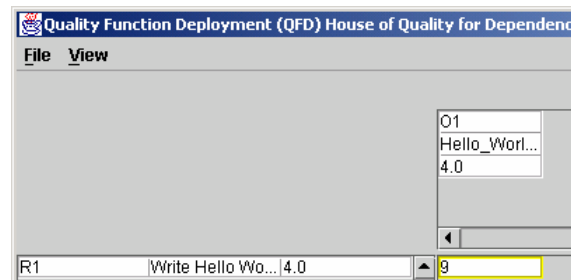


Figure 29 Dependency Threshold Result

With hundreds of atomic components, this view can be used to focus the design, for instance, on the “Top 10” components for a given dependency. Mean and standard deviation calculations are implemented for the top and left components by the Java code in Figure 30.

```

// get the left mean
for (int i=0; i<left.getRowCount(); i++)
    leftSum += gdv.getValue(left.getValueAt(i,2));
leftMean = leftSum/left.getRowCount();
// get the left std
for (int i=0; i<left.getRowCount(); i++)
    leftSTD += gdv.square(gdv.getValue(left.getValueAt(i,2)))-gdv.square(leftMean);
leftSTD /= left.getRowCount()-1;
leftSTD = java.lang.Math.sqrt(leftSTD);

// get the top mean
for (int i=0; i<top.getRowCount(); i++)
    topSum += gdv.getValue(top.getValueAt(i,2));
topMean = topSum/top.getRowCount();
// get the top STD
for (int i=0; i<top.getRowCount(); i++)
    topSTD += gdv.square(gdv.getValue(top.getValueAt(i,2)))-gdv.square(topMean);
topSTD /= top.getRowCount()-1;
topSTD = java.lang.Math.sqrt(topSTD);
double leftDecisionValue = leftMean+(nValue*leftSTD);
double topDecisionValue = topMean+(nValue*topSTD);

```

Figure 30 Dependency Threshold Calculations Code

CASES uses these values (e.g. leftMean, leftSTD, topMean, and topSTD) to reduce the view of the matrix based upon the decision values calculated (e.g. leftDecisionValue and topDecisionValue) with the user's input.

3. Component Trace

Once a matrix has been trimmed using the Dependency Threshold View, the stakeholder might be concerned with the components that influence a particular component (or the components influenced by a particular component). Consider the weighted digraph in Figure 31, which represents a simplistic process of requirements, specifications (consisting of operators "O" and data stream edges "E"), and code modules.

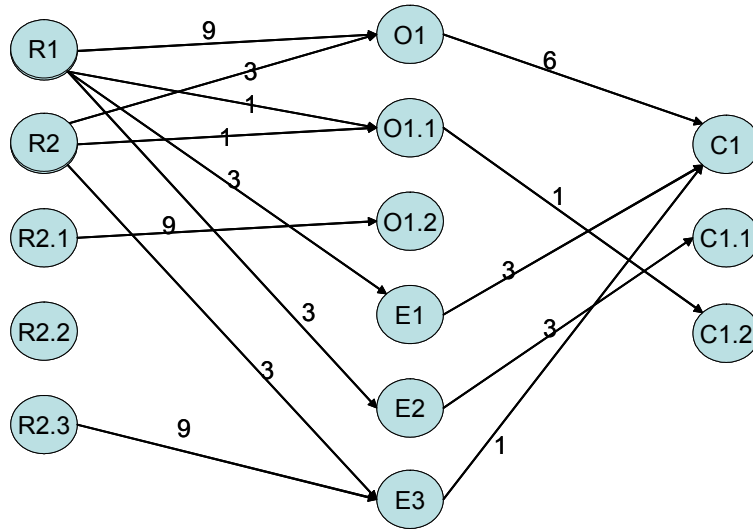


Figure 31 Weighted Digraph Example

Now recall the Dependency Threshold View (Figure 29) that identified requirement R1 and specification O1 as the most important components (from a “risk” perspective). Suppose having identified R1 as a component of interest, we now want to view the “knock-on” effect of potential changes to R1. We would obtain such a view by performing a Component Trace on component R1.

4. Component Trace Example

Figure 32 illustrates the Component Trace Dialog. This dialog gathers input information from the user to conduct a Component Trace on a row (e.g. “R1”) or column with a specified filter value (e.g. 3).

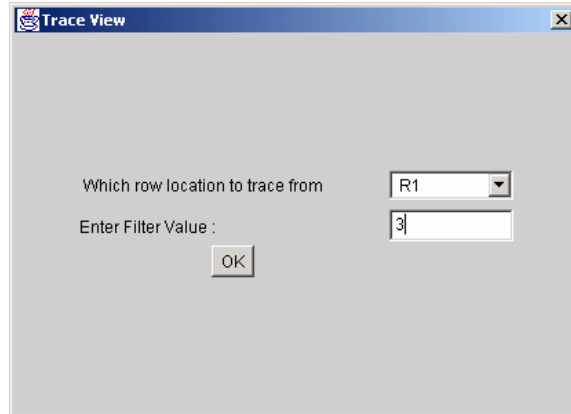


Figure 32 Component Trace Dialog Window

Upon execution of the trace, the weighted diagram (see Figure 31) is pruned to the induced subgraph shown in Figure 33. Notice that this subgraph could be further refined with a filter value of six (instead of three), which would prune this subgraph a new subgraph consisting only of R1, O1, and C1 (and the two steps).

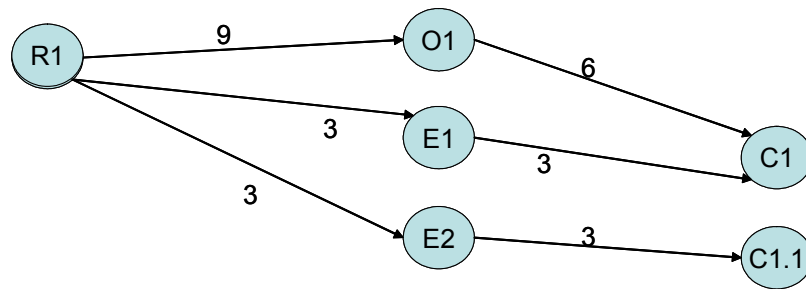


Figure 33 Component Trace Example

The two QFD matrices in Figure 34 are an equivalent representation of the induced subgraph in Figure 33. The top matrix represents the diagram of requirements (R1) to specifications (O1, E1, and E2) and the bottom matrix represents the diagram of specifications to code (C1 and C1.1).

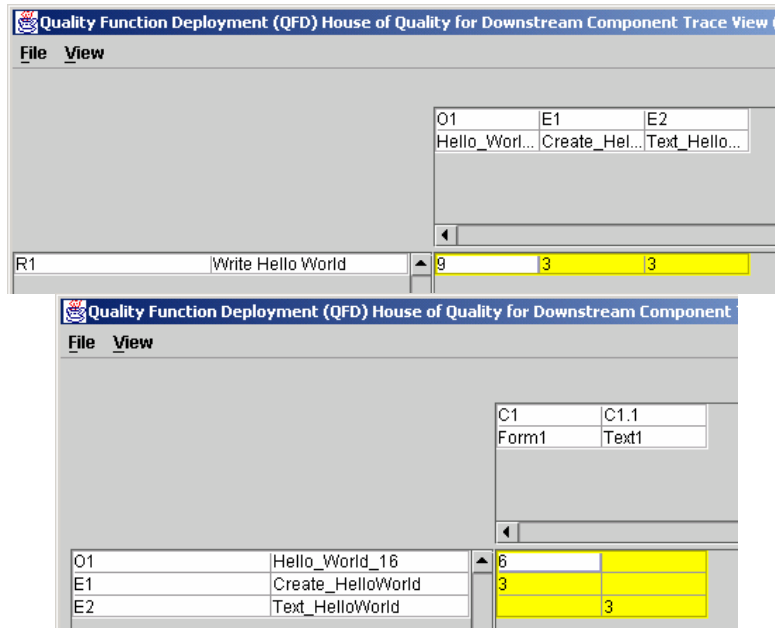


Figure 34 Component Trace Results

This component trace is currently limited to a single specified version and variant number. Future version of CASES should provide the stakeholder with multiple alternatives for tracing components including the following: ‘n’ version depth (where ‘n’ is a natural number of one or more versions) and exhaustive version depth (where ‘n’ equals the number of existing versions).

Component trace calculations are implemented by the code in Figure 35.

```

if( j==0 ){ // trace on a column
    TraceViewGUI traceView = new TraceViewGUI(dialog, topElements);
    viewValue = Double.parseDouble(traceView.getFilterValue());
    colSelected = traceView.getColumnSelected();
    new TraceUpstream (projectName, houseVersion, topID, depIndex, viewValue, colSelected);
    new TraceDownstream (projectName, houseVersion, topID, numOfComponents, depIndex, viewValue, colSelected);
} else { // trace on a row
    TraceViewGUI traceView = new TraceViewGUI(dialog, leftElements, j);
    viewValue = Double.parseDouble(traceView.getFilterValue());
    colSelected = traceView.getColumnSelected();
    new TraceUpstream (projectName, houseVersion, leftID, depIndex, viewValue, colSelected);
    new TraceDownstream (projectName, houseVersion, leftID, numOfComponents, depIndex, viewValue, colSelected);
}

```

Figure 35 Component Trace Calculations Code

CASES can trace based upon a user-selected column or a row. CASES uses two recursive objects (TraceUpstream and TraceDownstream) to trace the components that

meet the users input for a filter value (e.g. viewValue). Notice that the current version of CASES uses the variable “houseVersion” to scope the trace of components. Future versions should allow the stakeholder to cycle through a range of houseVersions. Cycling through versions must be based upon the dependency origin version. Therefore, if there are versions 1.1, 2.2, and 2.3 with a dependency origin at version 2.2, then the implementation must cycle through where 2.2 passes the “head” atomic components to the version 1.1 component trace (traceUpstream) and the “tail” atomic components are passed to the version 2.3 component trace (traceDownstream).

E. CASES AND OTHER TOOLS

CASES provides the stakeholder with a method to import data from other tools. A user can choose to import a CSV file for a component or a step. Component imports provide the user with a mechanism to populate the component artifacts within the QFD matrices as well as a way to populate dependency values. Step imports provide a mechanism to populate the correlation matrix between two components. The imported data must be of the proper dimensions (e.g. if component A has ‘m’ atomic components and component B has ‘n’ atomic components then the step import must contain ‘m x n’ correlation values).

Figure 36 is a dialog window that confirms whether the user desires to import dependency values or accept the default value (e.g. the value entered in dependency dialog). Importing dependency values will set the origin for that dependency based on the version number selected for import.

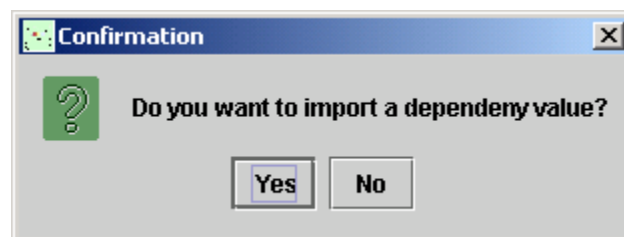


Figure 36 Import Dependency Value Confirmation

Figure 37 provides the user with a dependency listing from which the user can select the appropriate dependency to apply the imported values. The imported values will change the data specific to that component, dependency, and version only. Therefore, any generic data for other dependencies must be imported prior to a dependency value import.

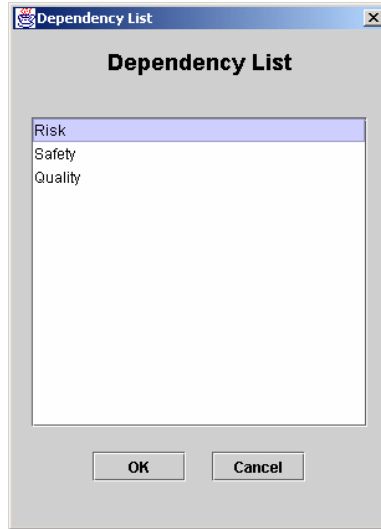


Figure 37 Dependency List Dialog

Figure 38 provides the user with a column header listing from the imported file. This provides the user with a method for matching the dependency to a specific column without the concern for name matching. In this example, the file has a column header that matches the dependency name of "Risk". Again, the matching of these names is left to the discretion of the user.

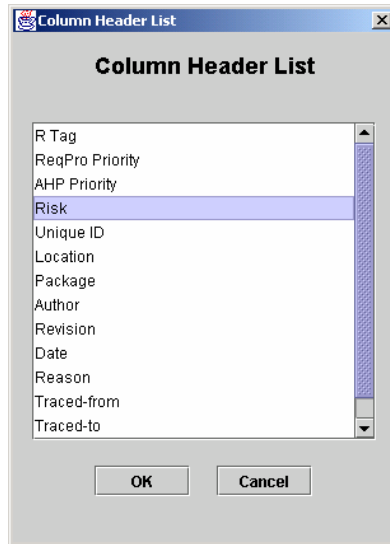


Figure 38 Column Header List Dialog

F. CHAPTER SUMMARY

This chapter presents key QFD concepts, views, and functionality, which are implemented within this thesis. CASES 2.0 implements the following QFD functionality:

- House of Quality using JTables.
- Automated upstream and downstream calculations.
- Storage for HOQ roof information including drivers and indicators.
- User-defined views including dependency threshold and component trace.
- Capabilities for importing CSV files from other tools into CASES.

Additionally, this chapter points out the mathematics behind, code examples of, and limitations in the implementation of these concepts, views, and functionality.

IV. CASES EXTENSIONS

A. INTRODUCTION

During the inception phase of this endeavor, it was determined that an improved architecture was warranted to improve the system structure and modularity, to make the system as a whole more understandable, and to provide a framework from which extensions could be added. In order to extend CASES (for this version and any future version), the new architecture must support the decomposition of the system, resources, and responsibilities.

This chapter provides an overview of the old architecture, shows the modifications and extensions to create the new architecture, and highlights some remaining limitations in the current implementation. Additionally, this chapter elaborates on and ties the java code in Appendix B to the architectural descriptions of the following packages: Project Schema GUI, QFD deployment, QFD views, and data imports.

B. CASES 1.1 ARCHITECTURE

Recall from Chapter II that CASES 1.1 consists of a single package designed with thirty-five java classes that are launched by the *CasesFrame.java* file. Figure 39 provides an illustration of the architecture of CASES 1.1, which consists of eighteen major classes, their primary methods, and relationships in the application (note: the eight minor classes and nine interfaces are not displayed in order highlight the central functionality). Unfortunately, this single monolithic architecture limits the extensibility of CASES. A multi-dimensional architecture increases code understandability and maintainability.

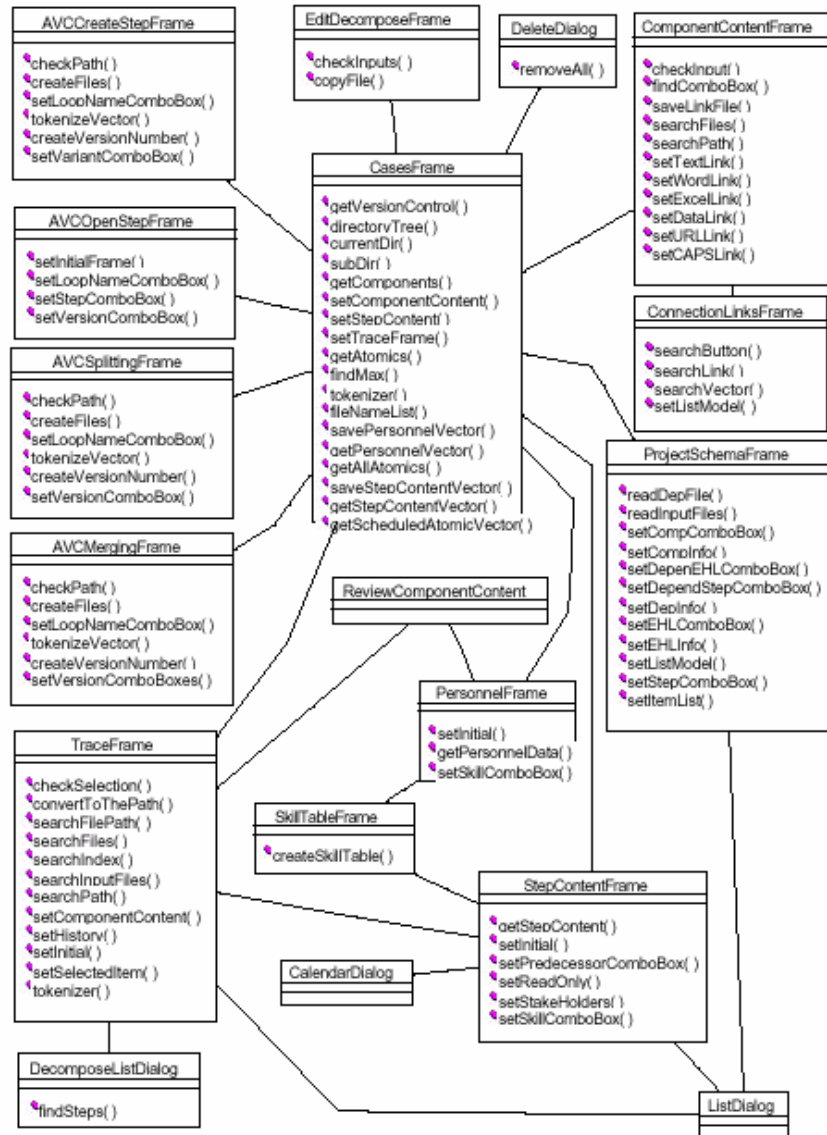


Figure 39 CASES 1.1 Class Diagram (from [LEHC99])

The old architecture was improved through the following:

- Updating the JDK from version 1.1.7 to 1.3.1,
- Designing and implementing a modular software architecture, and
- Eliminating the dependency on the Visual Café development environment.

The rationale for eliminating the dependency on the Visual Café development environment includes the following: Visual Café doesn't recognize hand-edited Java code, it requires the use of Webgain® code in several situations, and Webgain has been in the process of selling all of its intellectual property (therefore, the future of Visual Café is uncertain).

To illustrate the improvements in architecture, Table 4 provides the name and a short description of some standard software metrics used to compare the old and new architectures.

Abbreviation	Name	Description
LOC	Lines Of Code	This is the traditional measure of size. It counts the number of code lines. Documentation and implementation comments as well as blank lines are not counted.
NOA	Number Of Attributes	This is the count of the number of attributes.
NOC	Number Of Classes	This is the count of the number of classes.
NOIS	Number Of Import Statements	This is the count of the number of imported packages/classes.
NOM	Number Of Members	This is the count of the number of members, i.e. attributes and operations.
NOO	Number Of Operations	This is the count of the number of operations.

Table 4 Software Metric Description

Table 5 provides a short list of software metrics for the two versions of CASES' ProjectSchemaFrame (PSF) class as an example of architectural improvements contained in CASES 2.0.

PSF	LOC	NOA	NOC	NOIS	NOM	NOO
V1.1	1211	83	4	6	128	45
V2.0	986	31	1	18	92	61

Table 5 Basic Software Metrics

Note that the LOC, NOA, NOC, and NOM have been significantly reduced (18% or more) for the new version. This was accomplished by separating the panel functionality of PSF into four classes and removing Visual Café required internal classes (e.g. SymAction, SymItem, and SymMouse) with standard Java event listeners (e.g.

ActionListener, ItemListener, MouseListener) (see Figure 40). This composition approach creates highly cohesive objects that increase understandability and decrease the troubleshooting effort.

Also note that while NOO and NOIS have increased, this was necessary to provide operations to the GUI draw panel and to import the classes based on the three-layered package architecture (see Section C.1.).

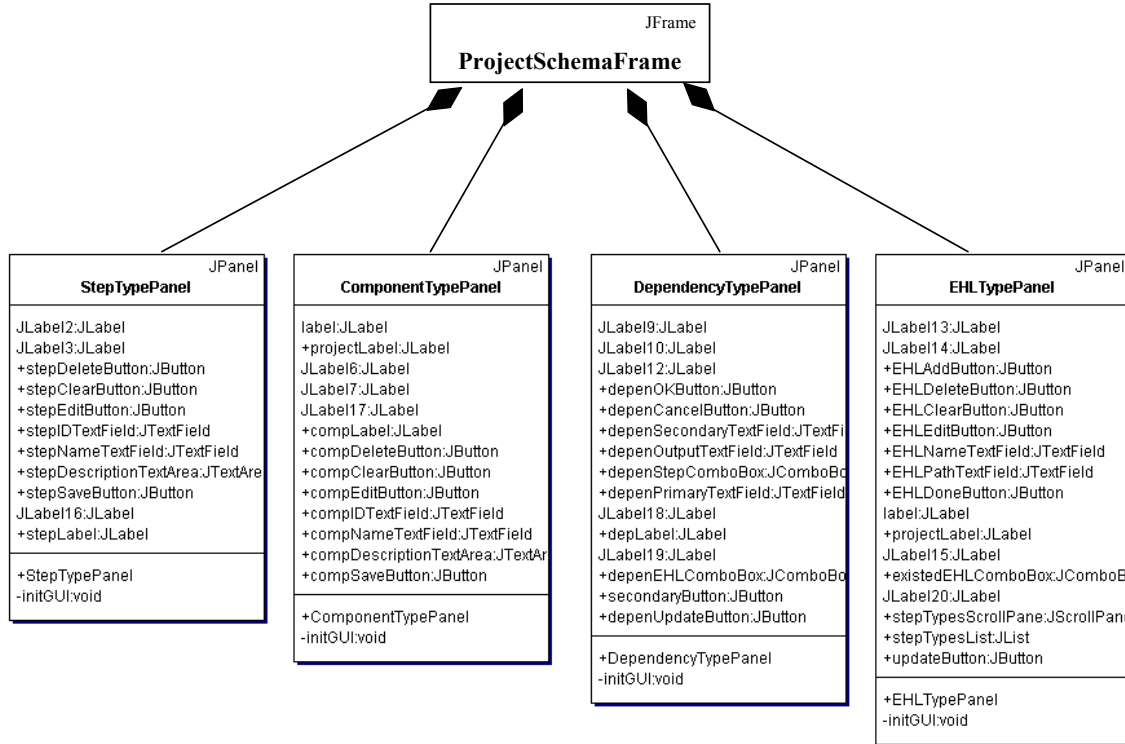


Figure 40 Composition of CASES 2.0 ProjectSchemaFrame

C. CASES 2.0 ARCHITECTURE

1. Layered Package Architecture

Figure 41 illustrates the top-level design of CASES 2.0, which consists of six packages and twenty-four classes. The *CasesFrame.java* class remains the initial launching point for CASES. CASESv2 packages are organized into three layers: subsystem, message, and responsibility. The subsystem layer contains a representation to implement the technical infrastructure that supports the CASES requirements. CASES is organized into coherent packages based upon system decomposition. The subsystem layer consists of GUI, Requirements, QFD, and JobSchedule packages. The message

layer contains the design details that enable each object to communicate effectively internally or externally. The message layer consists of interfaces, images, and data packages. The responsibility layer contains the data structure and algorithmic design for certain objects. The responsibility layer consists of all other packages (e.g. avc, psf, util, etc.) not contained in the subsystem and message layers. This three-layered design improves the system's decomposability, composability, understandability, and continuity. Additionally, this design provides robustness and reduces the propagation of unintended behavior if an error occurs in a particular module.

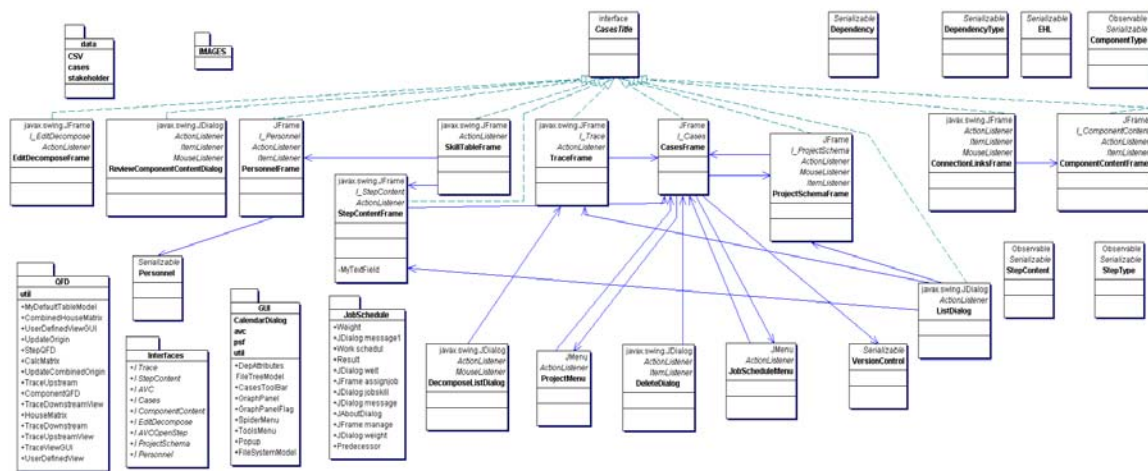


Figure 41 Cases Package Diagram

a. Subsystem Layer

The subsystem layer consists of GUI (see Figure 42), Requirements (a directory for UML requirement diagrams), QFD (see Figure 43), and JobSchedule packages (see Figure 46).

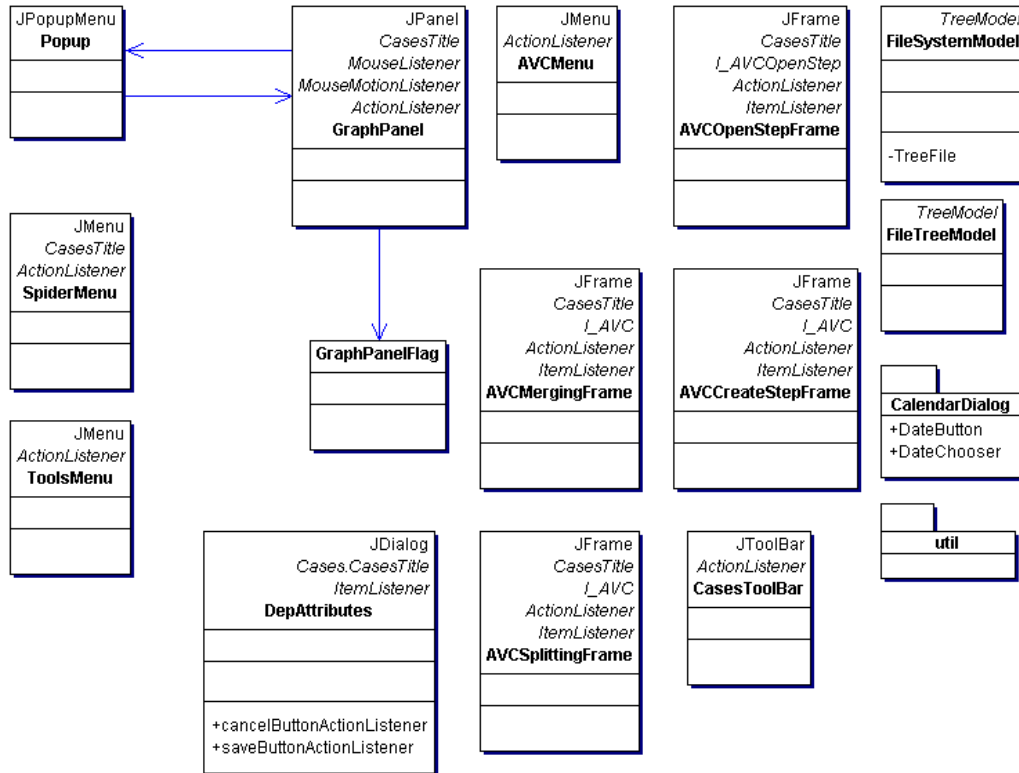


Figure 42 Cases.GUI Package Diagram

(1) GUI Subsystem Layer. The GUI subsystem consists of responsibility packages and many Java Swing extended classes (JFrame, JDialog, JMenu, JPopupMenu, JToolBar, and TreeModel). This subsystem provides the functionality for CASES to interact with users through the CASES' GUI. Examples of this functionality include popup menu, draw pane, tool bar, and automatic version control management.

The *GraphPanel* class (see Appendix B Section B.5) is the heart of the GUI subsystem. It provides the user with a canvas to draw the project schema and manages the graphical representation that communicates changes to other classes. It depends on the *GraphPanelFlag* class (see Appendix B Section B.6) to manage the states of various buttons (e.g. drawing a component, drawing a step, and so on) within the *CasesToolBar* (see Appendix B Section B.1). The *GraphPanel* class and *Popup* class (see Appendix B Section B.7) are interdependent for gathering information requested by the user for viewing or importing QFD matrices.

The *DepAttributes* class (see Appendix B Section B.2) extends *JDialog* for gathering and managing dependency information (e.g. name, description, type, origin, default value, etc.). *DepAttributes* has dependency properties with public methods for modifying the object's state.

The *FileTreeModel* class (see Appendix B Section B.4) and *FileSystemModel* class (see Appendix B Section B.3) provide the user with a static view of the CASES file directory structure. Additional work is required for these classes to capture dynamic changes to and user interactions with the file directory structure.

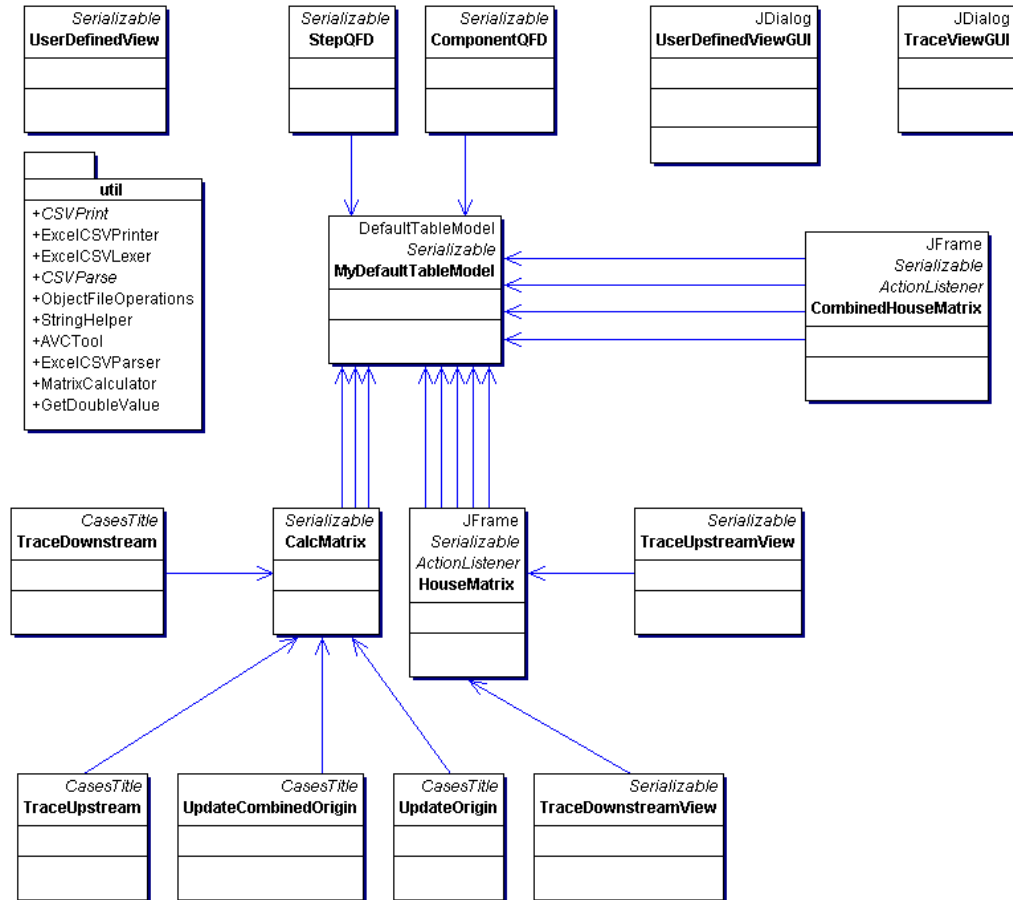


Figure 43 Cases.QFD Package Diagram

(2) QFD Subsystem. The QFD subsystem consists of QFD related classes and other responsibility packages. This subsystem provides the functionality for the house of quality, step QFD class composition, component QFD class composition, trace views, automated calculations, and much more.

At the heart of the QFD subsystem is the *MyDefaultTableModel* class (see Appendix B Section F.5) that extends the standard *DefaultTableModel* class and implements *Serializable*. *Serializable* is used through out the architecture to save and retrieve classes and their state through input/output streaming operations. The *MyDefaultTableModel* class includes an *origin* property that is used by the method *isCellEditable* to allow a user to edit cells within an origin's *JTable* or prevent a user from editing cells for non-origin *JTables*.

The *HouseMatrix* class (see Appendix B Section F.4) and *CombinedHouseMatrix* class (see Appendix B Section F.2) classes provide the visual representation of the House of Quality for the user. *HouseMatrix* allows for a single secondary-input step where *CombinedHouseMatrix* allows for multiple secondary-input steps for a specific component. The *TraceUpstreamView* class (see Appendix B Section F.10) and *TraceDownstreamView* class (see Appendix B Section F.8) are dependent upon *HouseMatrix* for displaying trace view results.

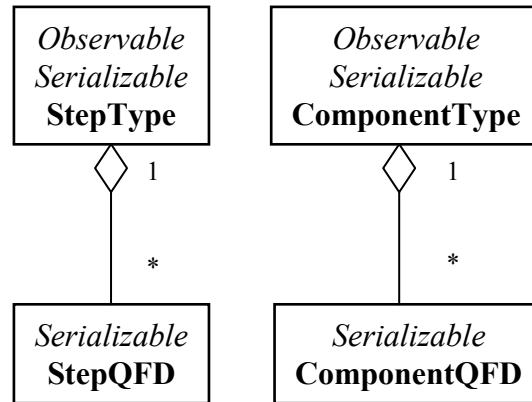


Figure 44 Aggregation of Types

Figure 44 shows the class diagram for extending the *StepType* class and the *ComponentType* class from the original architecture. QFD properties may be aggregated into the original types to capture *JTable* state information. Each type can have multiple QFD types (e.g. *StepQFD* or *ComponentQFD*) depending on the number of versions. While building the project schema, initially no versions exist; therefore, the types (*StepType* and *ComponentType*) will exist without the QFD extensions. Additionally, the original architecture types (*StepType* and *ComponentType*) were

extended using *Observable* in the model-view paradigm (discussed later in Section C.2 of this Chapter).

The *CalcMatrix* class (see Appendix B Section H.1) implements the individual QFD Deployment calculations defined in Chapter III. Figure 45 provides a sequence diagram of temporal events between *HouseMatrix* and *CalcMatrix*. The *UpdateOrigin* class (see Appendix B Section H.13) and *UpdateCombinedOrigin* class (see Appendix B Section H.12) implement the synchronized QFD Deployment calculations and are dependent upon *CalcMatrix* for individual calculations. These classes form the HOQ matrices in the order dependent upon the dependency origin and version. They communicate with *CalcMatrix* such that the class can determine when to call methods *forwardCalculation* or *reverseCalculation* (see Figure 45).

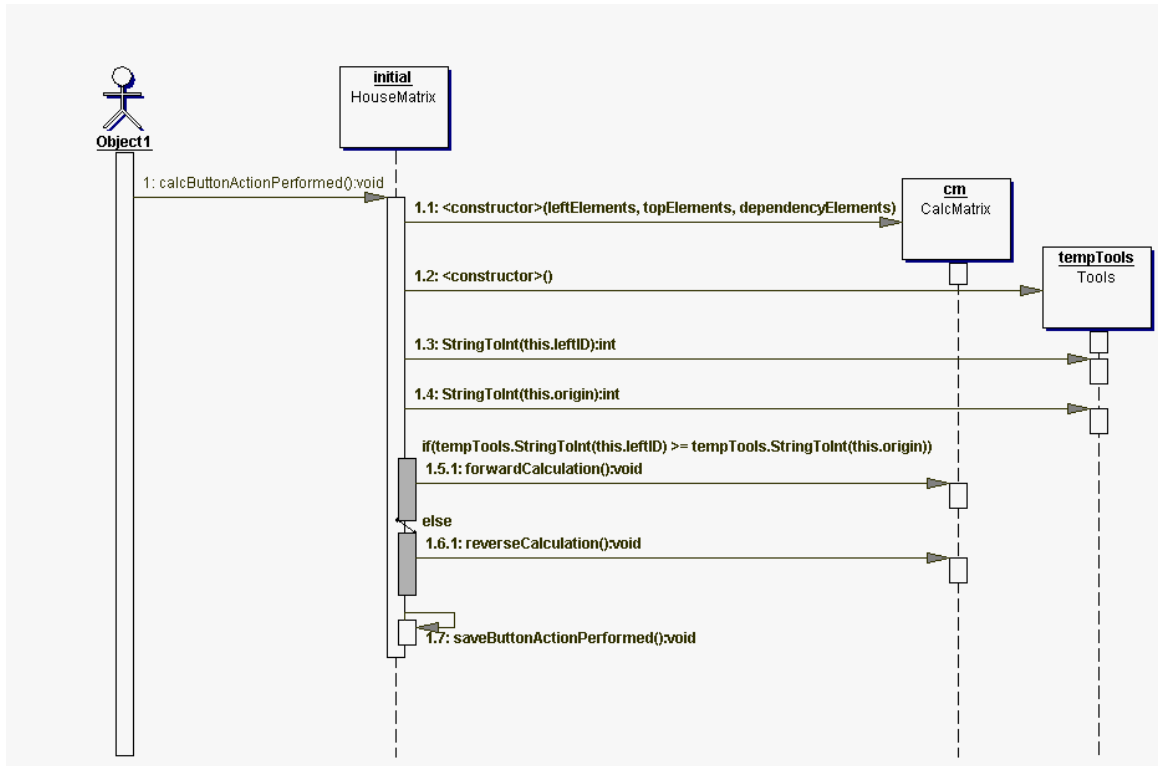


Figure 45 Individual Calculation Sequence Diagram

(3) JobSchedule Subsystem. The JobSchedule subsystem consists of job, skill, and schedule related classes and other responsibility packages. This subsystem provides the functionality for assigning jobs, classifying job skills, scheduling

jobs, and managing jobs. These classes existed in the CASES 1.1; but, CASES 2.0 organizes them into a single coherent package in a modified form with all dependency on Webgain Visual Café libraries removed (e.g. JButtonGroupPanel, Calendar, and GridBagConstraintsD) to better promote understandability and extensibility.

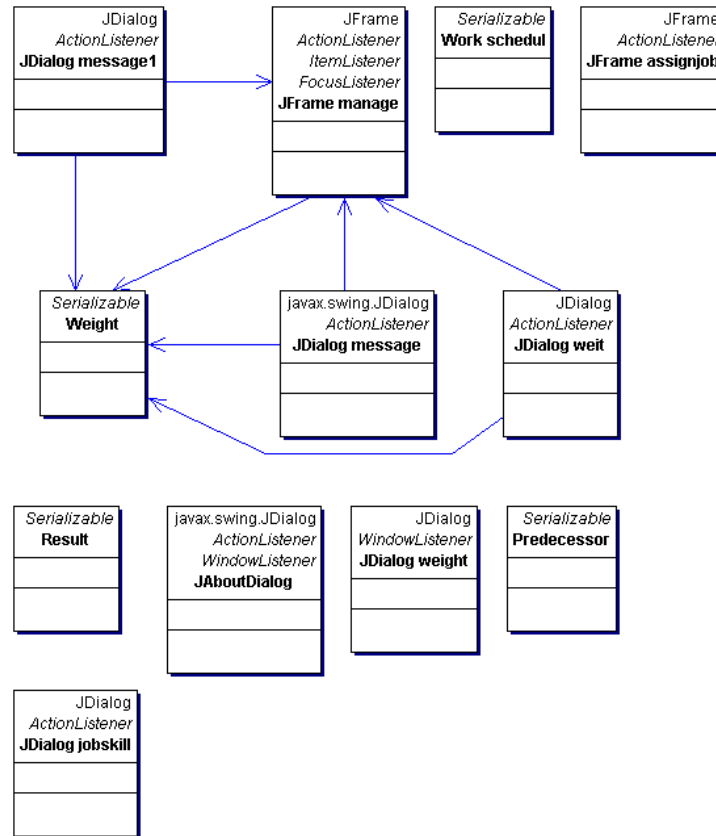


Figure 46 Cases.JobSchedule Package Diagram

b. *Message Layer*

The message layer consists of interfaces, images, and data packages. All interfaces (see Figure 47) that were created by [LEHC99] remain unchanged and are organized within this package.

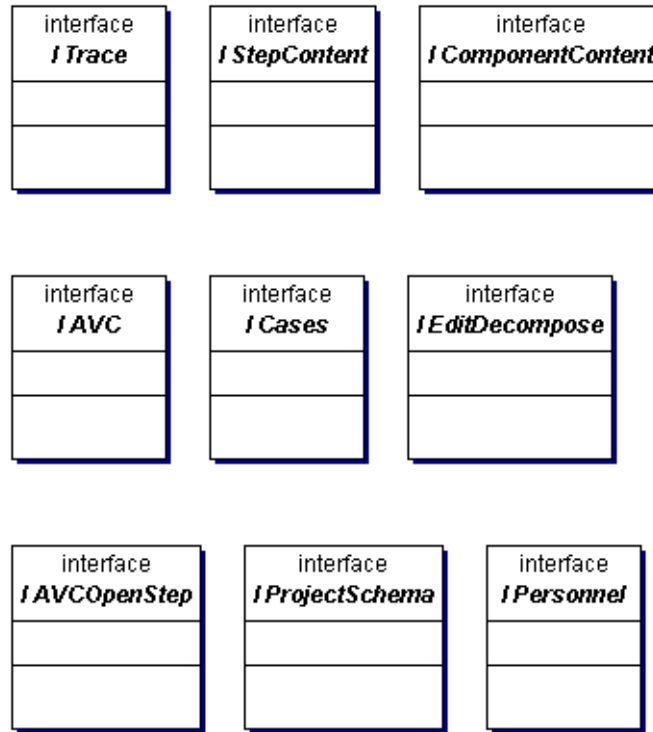


Figure 47 Cases.Interfaces Package Diagram

Table 6 provides a summary of interface information. The extensions in CASESv2 use several of these interfaces and leave the remainder unchanged. However, as future research, additional extensions could be added that make better use of the remaining interfaces provided by [LEHC99].

Interface	Implemented By	Functionality
I_Cases	CasesFrame	Interface to create the main frame for CASES.
I_AVC	AVCCreateStepFrame AVCSplittingFrame AVCMergingFrame	Interface to create a new step version. Interface to split a step version. Interface to merge step versions.
I_AVCOpenStep	AVCOpenStepFrame	Interface to open a step version.
I_ComponentContent	ComponentContentFrame	Interface to create/view/edit/delete link files in the Component Content directory.
I_EditDecompose	EditDecomposeFrame	Interface to edit/decompose the current step.
I_Personnel	PersonnelFrame	Interface to create/edit/view a personnel object.
I_ProjectSchema	ProjectSchemaFrame	Interface to create/edit step.cfg, component.cfg, loop.cfg, and dependency.cfg.
I_Trace	TraceFrame	Interface to view the selected component.

Table 6 CASES Interfaces (after [LEHC99])

The images layer provides a central repository for locating and calling the CASES icon (e.g. Cases.ico) and graphic images needed by CASES (see Table 7). Most of these images provide graphics for CASES toolbar buttons.

File Name	Description
Cases.gif	Cases application logo in gif format.
Cases.wmp	Cases application logo in windows media player format.
Cases.ico	Cases Icon used by windows and dialogs.
Component.gif	Cases Toolbar graphic for component button.
Delete.gif	Cases Toolbar graphic for delete button.
Dep.gif	Cases Toolbar graphic for dependency button.
Lock.gif	Cases Toolbar graphic for PSF lock button.
Redo.gif	Cases Toolbar graphic for redo button (provided for future improvements).
Select.gif	Cases Toolbar graphic for select button.
Square.gif	Cases Toolbar graphic for square button background.
Step.gif	Cases Toolbar graphic for step button.
Synchronize.gif	Cases Toolbar graphic for synchronize button.
Undo.gif	Cases Toolbar graphic for undo button (provided for future improvements).

Table 7 Image Package

Similarly, the data package is a central repository for project data, stakeholder information, and CSV files required by CASES (see Table 8). The cases directory is the location CASES manages projects and their data. The CSV directory is a central location CASES will search by default for CSV files. The stakeholder directory is the location CASES will manage personnel data.

Directory Name	Description
Cases	The cases directory is the location from which CASES manages projects and project data.
CSV	The CSV directory is the default location from which CASES browses for CSV files.
Stakeholder	The stakeholder directory is the location from which CASES manages personnel data.

Table 8 Data Package

c. Responsibility Layer

The responsibility layer consists of all other packages (e.g. avc, psf, util, etc.) not contained in the subsystem and message layers. These low level packages have specific responsibility to CASES or the subsystems. Table 9 provides a summary of these packages.

Package	Responsibility	Subsystem	Remarks
avc	Creating, merging, splitting, and viewing of version information.	GUI	Location for Automatic Version Control (AVC) classes for most version and variant operations.
psf	Component, step, dependency, and evolution hypergraph management.	GUI	Location of the ProjectSchemaFrame composite JPanel classes.
util	Next and previous version retrieval and other tools.	GUI	Location of additional AVC operation and utilities.
CalendarDialog	Provide date chooser for the user.	GUI	Location of task and calendar operations.
util	Matrix, mathematic, and CSV file operations.	QFD	Location of operations and utilities required by the QFD subsystem.

Table 9 Responsibility Package Layer

2. Patterns

In addition to a layered package architecture, CASES 2.0 employs the Model-View-Controller (MVC) architecture (see Figure 48) using the Java class *Observable* and Java interface *Observer* to enable CASES to manage software evolution artifacts (specifically, component and step types). The *TypeController* (*ProjectSchemaFrame*) accepts user input and modifies artifact types. The Type classes are *Observable* objects that act as CASES' models. When the *TypeController* modifies a type, the type notifies the *GraphPanelView* with the modified information. This model provides a foundation for additional views.

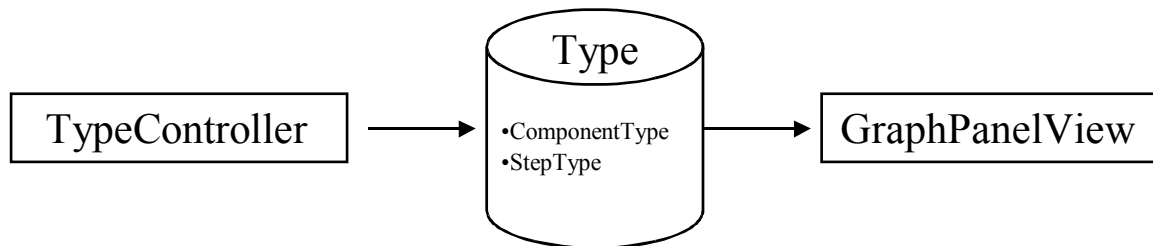


Figure 48 CASES Application MVC Architecture

For example, when a user draws a component on the draw pane (*GraphPanelView*) a *componentType* is created. Additionally, as more components are

created and connected with steps the draw pane updates or creates the information to reflect the drawing. All information is *serializable*; therefore, it is streamed to a configuration file on the local system. Components are stored in the *component.cfg* file and Steps are stored in the *step.cfg* file. If the user changes the name of that component (e.g. “requirements”) or a step (e.g. “generate code”) through the *ProjectSchemaFrame* – the *TypeController* will automatically update the draw pane to reflect the new name. Therefore, the view will remain consistent with the data stored in the configuration files.

D. DATA IMPORT

Data import is a critical function of CASES. Any computer-aided tool must remove or reduce the burden of data entry when information is available from other software resources. CASES accomplishes this task

by allowing the user to import CSV files (see Figure 49). CASES uses the class *ExcelCSVLexer* (See Appendix B Section I.2) which is a client dependent on a supplier file (e.g. *file.csv*).

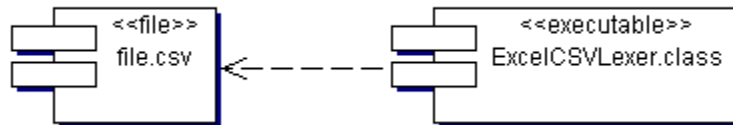


Figure 49 UML for Current Import Data Architecture

A better implementation for future versions of CASES is an abstract class to interface with other software tools (see Figure 50). With an interface, concrete classes may be specialized to import data from specific software tools and formats (e.g. Microsoft Excel CSV file imports). This would provide a more robust architecture with flexibility to support future file and data formats and middleware implementations.

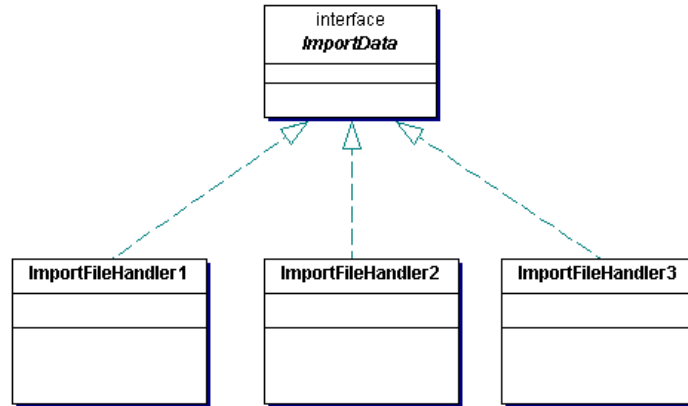


Figure 50 Recommended Import Data Architecture

E. LIMITATIONS OF THE CURRENT IMPLEMENTATION

CASESv2.0 has the following limitations that may require future modification:

- Locked Project Schema.
- Single Version Component Trace.
- Bounded Arrays for GraphPanel.
- Inefficient data representation.

The “locked” project schema was discussed in Chapter III, which stated that the project schema must be locked before the automatic version control (AVC) function can be utilized. This limitation stems from CASES 1.1 and has migrated into this implementation because the 2.0 extensions use the AVC functionality. Future work should provide a project schema environment in which the user can dynamically change the structure of the project schema at any time. Such an addition may result in a modified or different implementation of the relational hypergraph. CASES 1.1 uses EHL objects (see Appendix B Section A.10) to manage version numbers. The recommended improvement would be to use a MVC pattern, which dynamically updates EHL object clients.

The single version component trace can be improved by adding an implementation that allows the user to traverse all versions or traverse a specific portion of the development effort. The new implementation should use the current

implementation of the *TraceUpstream* and *TraceDownstream* classes, which take a single version as a parameter. The new implementation could query the user for a range of valid versions to traverse and pass the range values (one at a time) to the current implementation using the pseudo-code in Figure 51.

```
if (range version < origin version) then
    TraceUpstream (range version);
else if (range version > origin version) then
    TraceDownstream (range version);
else if (range version = origin version) then
    TraceUpstream (range version);
    TraceDownstream (range version);
```

Figure 51 Trace Pseudo Code

The trace will be upstream for versions less than the origin version and downstream for versions greater than the origin version. If the range version is equal to the origin version, then both upstream and downstream must be called because an upstream trace must occur from the dependency component to the start node and a downstream trace to the end node.

The GraphPanel was prototyped with bounded arrays with the global variable in *CasesTitle* (see Appendix Section A.2). Figure 52 provides an extraction from *CasesTitle*. MAXDEPENDENCIES and MAXCOMPONENTS variables are used to bound dependency, component and step arrays. A better implementation would be to use unbounded data structures and extend the stepType and componentType classes to include information required by GraphPanel. CASES data should be represented in a single efficient form and the MVC pattern used to update all client objects. This approach is used for the component and step names, but should be consistent throughout the applications for CASES data required by the GUI subsystem.

```
// data is used for GraphPanel
static final int MAXCOMPONENTS = 10;
static final int MAXDEPENDENCIES = 10;
static final int MAX = MAXCOMPONENTS + 1;
static final int COMPONENTSIZE = 50;
static final int NAMESIZE = 50;
static final int COMPONENTRADIX = 35;
static final int DEFAULT_WEIGHT = 50;
```

Figure 52 Extract from CasesTitle of GraphPanel Limitations

CASES 2.0 QFD data are currently represented with JTables. This representation might not provide the performance required for large software development projects with many different types of components, large numbers of artifacts, and large numbers of dependencies. Additionally, the QFD matrices can be very large, but are generally very sparse. This disparity between QFD cells used against cells unused leads to an inefficient representation. Additional research is needed to evaluate the best representation for QFD data.

F. SUMMARY

CASES' architecture has been extended and improved from a single monolithic design to a three-layered design. The GUI and QFD subsystems contain the bulk of the improved functionality. The GUI subsystem gives the user a practical method for creating and managing a project schema. The QFD subsystem gives the users a framework for creating, managing, and viewing dependency relationships between atomic components. The QFD subsystem has many powerful tools and utilities for automated calculations and specialized engineer views of critical data. Additionally, the model-view-controller pattern is implemented to provide a flexible architecture for the current and future views of the project schema. CASES imports data files (in CSV format) to reduce the user's data entry burden and to provide interoperability with external software development tools. While some improvements are necessary to eliminate some of the limitations of this software tool – the tool in its present form provides significant benefit in allowing software engineers to make better use of development information.

V. CONCLUSION

A. SUMMARY

U.S. industry and government become more reliant on software everyday; yet, the production of safe, reliable software, produced on-time and on-budget that meets the customer's requirements is an increasing challenge because of changing technology and increased complexity. This trend combined with the compelling research listed below provides the motivation for developing tools to support holistic software development:

- Holistic Framework for Software Evolution [PUET02, 03],
- Relational Hypergraph Software Evolution Model [HARN99a],
- CASES 1.1 [LEHC99], and
- Quality Function Deployment [CLAU88], [HAUS88], [COHE95], and [ZULT90, 92, 93].

Quality Function Deployment (QFD) is a requirements-based methodology used widely in the global product industry to improve process and product quality. However, the use of QFD as applied to software development has, to date, been very limited. Therefore, this thesis uses the QFD dependency relationships (generic in order to accommodate dependencies beyond “quality”), automation, and user-defined views as additions to the foundation work above.

This thesis is ground breaking in that for the first time, it merges a well-established product industry engineering practice (QFD) into the field of Software Engineering by integrating the QFD methodology into a software development evolution control system known as the Computer Aided Software Evolution System (CASES). This thesis extends CASES with a three-layered architecture, data import functionality, and the use of the model-view-control design pattern. These extensions improve maintainability and usability of this tool and provide a tool that directly supports holistic software development.

B. CONTRIBUTIONS

The most important contributions to the field of Software Engineering that this thesis provides is to accomplish the following:

- Embed QFD within the Relational Hypergraph Software Evolution Model,
- Provide engineering views of QFD dependencies, and
- Provide a stakeholder GUI.

Additional contributions include

- Designing and implementing a modular software architecture, and
- Eliminating the dependency on the Visual Café development environment.

These contributions allow a software engineer to: 1) Input, modify, and analyze dependency characteristics between software artifacts within a QFD framework; 2) Make decisions based upon views of dependency information; and 3) Design a custom software evolution model through the use of a GUI. These contributions lead to significant improvements in both the software development process and within software products. Perhaps more importantly, this contribution provides a research foundation upon which, software researchers will discover and leverage unrealized dependencies within software development.

C. FUTURE DIRECTIONS

CASES 2.0 is a research tool that supports the HFSE. As such, it is a work in progress. Research should continue to improve and extend CASES in order to demonstrate and validate future software development process/product improvement concepts. These future research efforts include both technical improvements (modifications to the existing CASES features) and conceptual improvements (major additions to the set of existing CASES features).

1. Technical

There are several technical modifications that are warranted to improve CASES and provide impetus for its adoption into the Software Engineering domain. The following provides a starting point for future research:

- Provide a GUI for stakeholders to conduct pairwise comparisons between atomic component artifacts to determine correlation values.
- Update the Project Schema implementation so that a project schema can be modified at any time during the project development cycle (eliminate the need to “lock” the schema -- as discussed in Chapters III and IV).
- Improve the Dependency Dialog to take advantage of different dependency types and ranges.
- Provide improved functionality so that a Component Trace can extend beyond a single version or be limited to a stakeholder selectable set of versions.
- Implement the “drivers” and “indicators” concept within the roof of the HOQ to produce improved QFD matrix views.
- Continue to improve CASES’ architecture, to include the following: interface improvement (build highly cohesive and loosely coupled objects) and unit testing (to remove defects).
- Provide additional documentation (e.g. Users Manual and Technical Manual) and improved Javadocs for future stakeholders of this software.
- Improve the HOQ view with synchronized scrolling between the top-most *JTable* and the correlation matrix as well as between left-most *JTable* and correlation matrix.

2. Conceptual

Conceptually, there are several CASES additions that would provide additional support for the HFSE. Some of the conceptual improvements include the following:

- Provide direct interoperability with OOMI IDE. CASES should be capable of directly importing generated Java translators from the OOMI IDE. These translators should be able to be directly mapped to particular artifacts and tools.
- Analyze the representation of information in the HFSE (e.g. tree structure, table structure, hypergraph, etc.). CASES should represent HFSE data in the most effective and efficient manner.
- Determine component dependencies creation path possibilities. CASES assumes a step-to-component path. Additionally, this path must begin at an initial node and end at a single node. Paths cannot have loops except at the bridge node, which joins the end node and the start node.
- Implement the CASES data import mechanisms using middleware. Such mechanisms will allow the dependency values and correlations values to be dynamically updated as software development artifacts are created, modified, or deleted.
- Implement sensitivity analysis statistical methods into the tool. Such methods will assist software engineers in quickly identifying those calculated dependency values that are most subject to change after minor perturbations in subjective correlation values.

D. CONCLUDING THOUGHTS

Embedding QFD into CASES provides partial solutions to the following problems: 1) the software crisis, 2) the increased demand for quality software, 3) inadequate customer-developer communication, and 4) the lack of tool support for building high assurance software systems. The demand and cost for quality software is ever increasing. The HFSE provides a conceptual framework and a software engineering process model to reduce the cost and increase the quality of software. As demonstrated by this thesis, a pragmatic implementation of the HFSE was established by embedding

the relevant portions of the QFD methodology into CASES to provide tool support for the HFSE.

The research presented in this thesis should be viewed as a first attempt at merging a well established product industry engineering practice into the field of Software Engineering by integrating the QFD methodology into a software development evolution control system (CASES). This research will lead to pragmatic approaches which can provide program managers and software engineers with the additional tools required to build safe, reliable software; produced on time and on budget that fully meets the needs of the Department of Defense and the nation.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. CASES 2.0 TUTORIAL

A. INSTALLATION INSTRUCTIONS

All required files for CASES 2.0 are provided on a CD-ROM, which contains the documentation, source code, and object code. Follow the MS Window environment instructions below to install CASES 2.0.

1. Put the CASESv2 CD-ROM into the CD-ROM drive.
2. Open a file manager and locate the *Cases2_Object_Code.zip* file on the CD ROM.
3. Open the *Cases2_Object_Code.zip* file and extract to the *C:* directory. The extraction will automatically create a *C:\Cases* directory and place the object code files within the directory.
4. Copy the Cases.bat file from the CD-ROM to the desktop. The MS DOS batch file contains two commands: *c:* and *java -classpath C:\cases; Cases.CasesFrame*.
5. Ensure that you have the Java development kit (JDK v.1.3.1 or later) installed on your system and the path to the application is set within the system environment settings (see Sun Java website for more information).
6. Double clicking the Cases.bat icon or typing the commands in Step 4 at the DOS prompt will start CASESv2. Enjoy!!!

The source code is available by extracting the *Cases2_Source_Code.zip* file to the *C:* directory. The extraction will automatically place the source code files in the *C:\Cases* directory.

B. MINIMUM SYSTEM REQUIREMENTS

The minimum system requirements for CASES 2.0 are provided in Table 10. The requirements are set to accommodate the complete installation of the Java Development Kit 1.3.1, CASES 2.0 source code, research documentation, and CASES 2.0 object code.

Cases 2.0	Solaris 8	Windows NT 4.0 SP 6, Window XP SP 1, Windows 2000 SP2	Red Hat Linux 6.2
Hardware System: Minimum	Ultra 10	350 MHz Pentium II	350 MHz Pentium II
Memory (RAM) Minimum	128 MB	128 MB	128 MB
Disk Space for Installation	380 MB	380 MB	380 MB

Table 10 Minimum System Requirements

Additionally, the installation will require a 2X CD-ROM drive. While not tested, CASES 2.0 should run on Solaris 8 and Red Hat Linux 6.2 operating systems, but will require the source code to be recompiled on those systems. Instructions are not provided for these environments.

C. HELLO WORLD EXAMPLE

This section provides a CASES v2.0 tutorial using the Hello World example presented in [PUET03]. The tutorial walks the reader through the creation of a project, development of a project schema, creation of versions and variants, creation of dependencies; importation of data; and execution of the QFD, SPIDER, and TOOL operations.

1. Creating or Loading a Project

Figure 53 illustrates the first step, which is to create a project through the *Project* menu. After selecting the create project menu item, a dialog window will request the project name (see Figure 54).

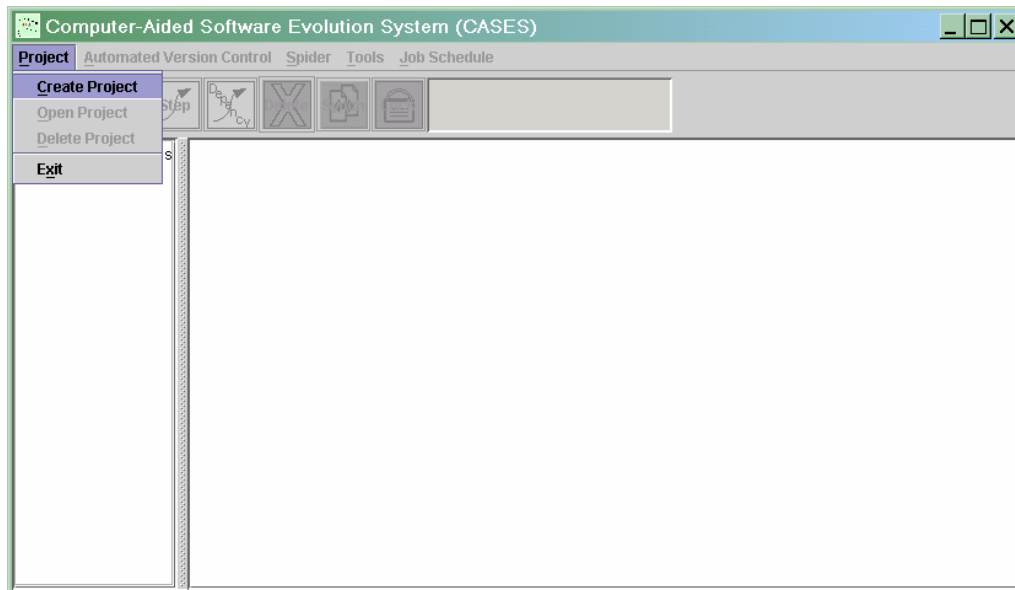


Figure 53 Create Project

For this example, the project name will be *HelloWorld*. After the project is created it will be saved to the *C:/CASES/data/cases/HelloWorld* directory. All projects are managed by CASES with separate project directories.



Figure 54 HelloWorld Project

A user also has the option of opening an existing project. Figure 55 shows the open project dialog window for retrieving projects from the *project* menu. This menu item is useful for switching between projects as well as loading a project upon starting CASES. The left most side of the CASES application GUI provides a listing of all projects within the project directory (*C:/Cases/data/cases*).

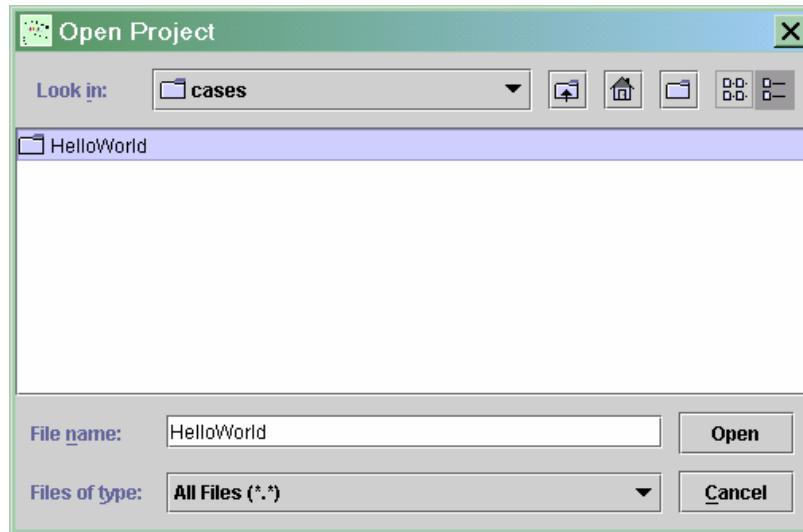


Figure 55 Open Project

2. Creating a Project Schema

After creating a new project, the software engineer creates a project schema that models their development effort. The project schema is an abstraction of the artifacts and activities (components and steps) of the actual development process the engineer uses. Figure 56 illustrates a completed project schema. A project schema is developed through a multi-step process. First, press the *Component* button on the toolbar. Then, add components to the draw panel by clicking locations on the draw canvas (note: the first Component drawn is the start node). Next, press the Step button on the toolbar. Then, add steps from one component to another by clicking the first component and then the next component (note: all paths must lead to a single end node and the end node may then be connected to the start node to symbolize additional software iterations). Finally, the project schema must be “locked” by pressing the Lock button on the toolbar. Locking the schema means that no additional high-level components or steps can be added. However, the existing components and steps may still be edited after locking the schema.

A popup menu is available by pressing the right mouse button on a component or step arrow. The popup menu provides the user with access to the properties of the objects in the project schema, to the QFD functionality in the schema, and to the import data functionality of the tool.

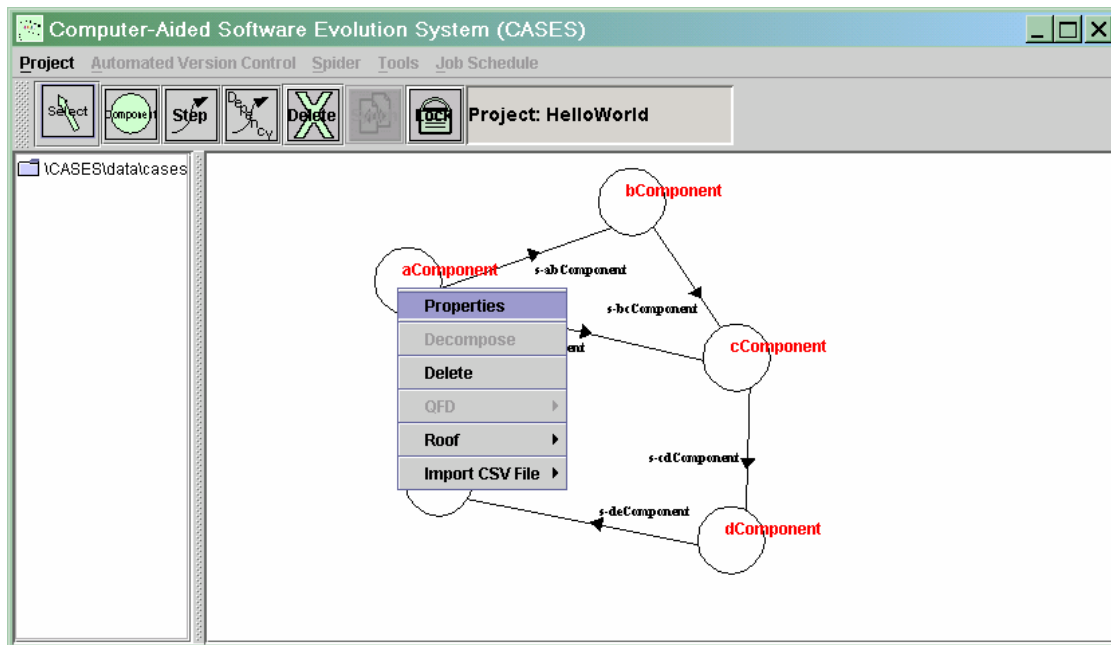


Figure 56 Popup Menu

If a user selects *Properties* from the popup menu then the Project Schema frame (see Figure 57) will become visible (note: double clicking a component or step arrow has the same result). The user can customize the component or step name and description within this frame and save the changes by pressing the Edit button. This example would change “aComponent” to “Reqs” with a description of “Customer Requirements”.

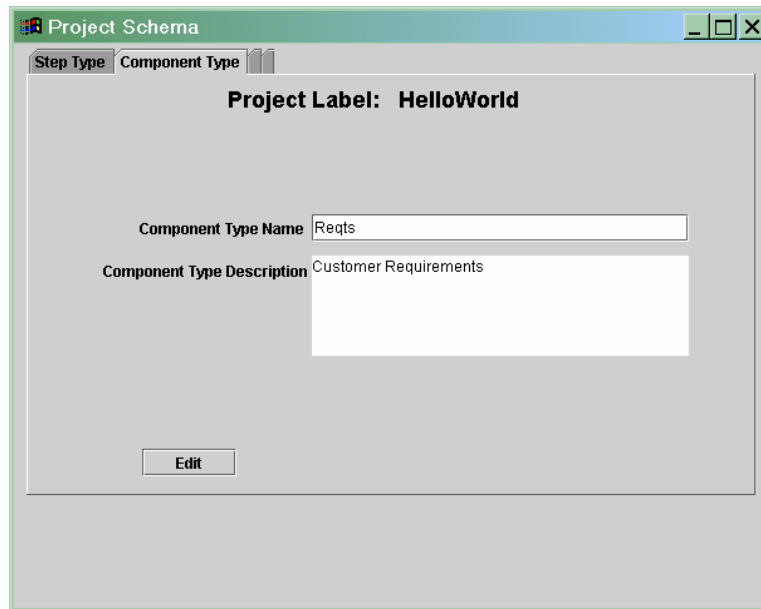


Figure 57 Component Project Schema

3. Creating Step Versions

Once all the components and steps are customized, the user will want to create a step “version.” This is accomplished by selecting the *Create Step Version* menu item from the *Automated Version Control* (AVC) menu (see Figure 58).

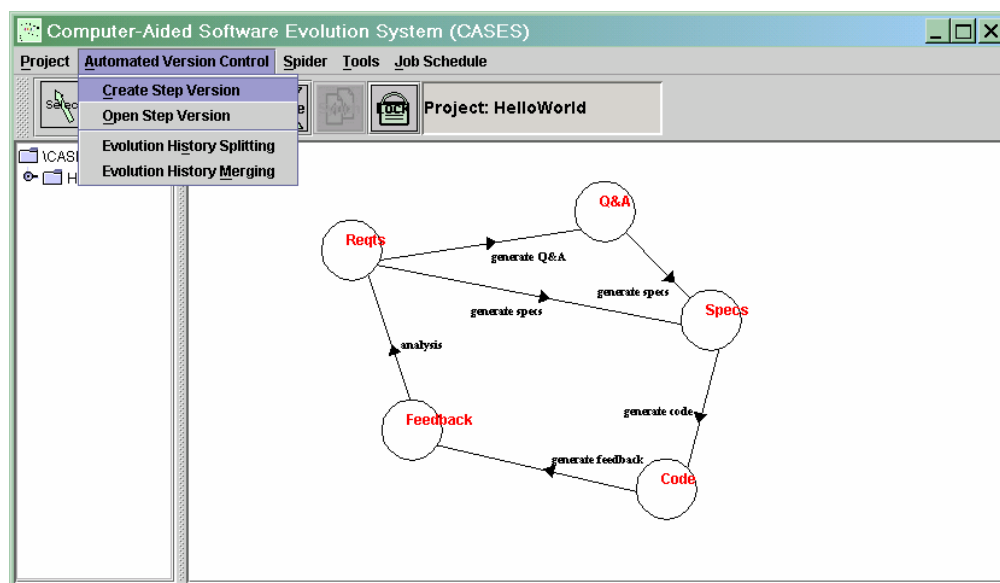


Figure 58 Automated Version Control

This will display the Create Step Version dialog window (see Figure 59). The user presses the *New Step Version* button and the appropriate version number will automatically be generated (based on the rules in [LEHC99]) and presented in the text box to the right of the button. The user can press the OK button to accept the new step version or the Cancel button to reject it (note: the initial step version is always 1.1). CASES assigns the new version number to all step and component objects within the project schema.

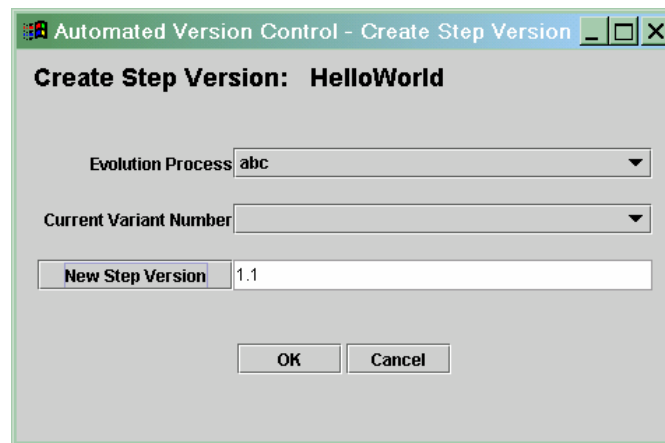


Figure 59 Create Step Version

After creating a step version, it must be opened to perform any SPIDER functions (note: this is a constraint of CASES 1.1). All SPIDER functions are based on the open step version. Figure 60 illustrates the dialog window resulting from selecting the *Open Step Version* menu item from the AVC menu. The Evolution Process is automatically set and generated by CASES 2.0. The Step Type is a step identifier, which represents internal information used by CASES (e.g. “s-acComponent” = a secondary input step from the “a” Component to the “c” Component). The user will be required to recall step identifiers if the user requires the use of the SPIDER functions. Future version of CASES should provide customized step names instead of step identifiers.



Figure 60 Open Step Version

Figure 61 illustrates the dialog window resulting from selecting the *Evolution History Splitting* menu item from the AVC menu. The user presses the *New Step Variant* button and the appropriate variant/version number will automatically be generated (based on the rules in [LEHC99]) and presented in the text box to the right of the button. The user can press the OK button to accept the new step variant or the Cancel button to reject it.

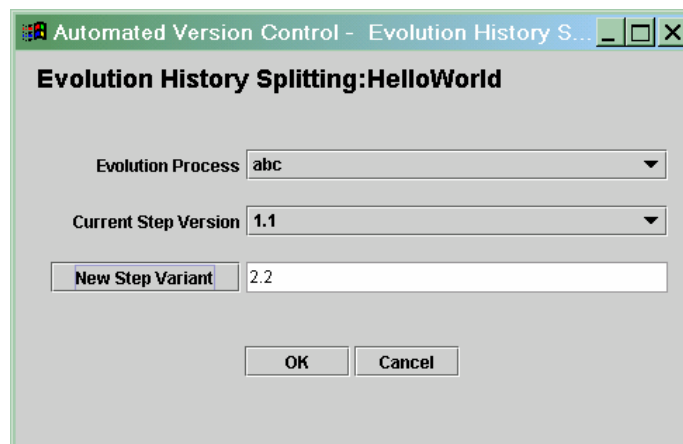


Figure 61 Splitting Step Version

Figure 62 illustrates the dialog window resulting from selecting the *Evolution History Merging* menu item from the AVC menu. The user selects the current step and merged step versions from the drop down list. The user must select from the variant type as either new or old. If the variant type is new, a new variant number will be generated;

else, an existing variant number will be used. The user presses the *New Step Version* button and the appropriate variant/version number will automatically be generated (based on the rules in [LEHC99]) and presented in the text box to the right of the button. The user can press the OK button to accept the new step variant or the Cancel button to reject it.

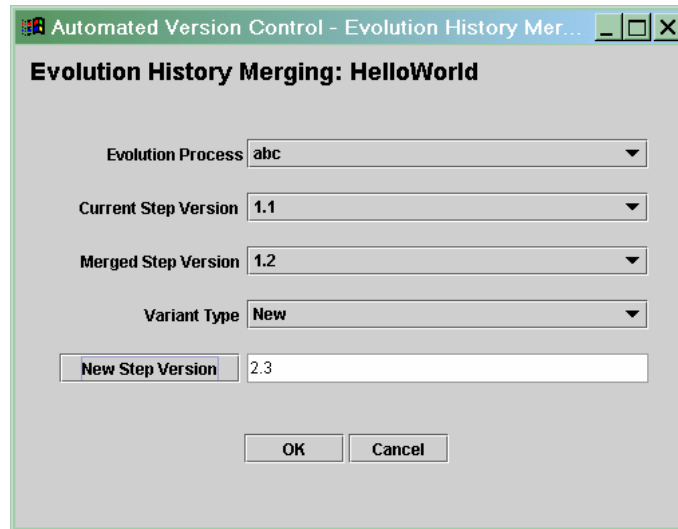


Figure 62 Merge Step Version

Recall, the Open Step Version functionality is required for SPIDER operations, but doesn't affect the CASES 2.0 QFD functionality. The CASES 2.0 project schema is an abstract visual representation of all existing versions. Therefore, a user can access dependencies, HOQ, roof and other QFD functionality for any of the created versions.

4. Creating Dependencies

Once versions have been created, the user is ready to create dependency. Figure 63 shows the dependency dialog, which is available by pressing the Dependency button on the CASES toolbar. The dependency dialog allows the user to provide information about a particular dependency (name, description, type, value range, default value, and origin). The origin must be located at a specific component and a specific version/variant. The version is initialized to 1.1, but can be modified by importing a CSV data file for a particular dependency (discussed later). For this example, a risk dependency is illustrated with an origin at the *Reqts* component.

The image shows a 'Dependency Attribute' dialog box. It has a title bar with the text 'Dependency Attribute' and a close button (X). The dialog contains the following fields and controls:

- Name:** A text input field containing the text 'Risk'.
- Description:** A text area containing the text 'Project Risk'.
- Type:** A dropdown menu with 'Risk' selected.
- Value Range:** A dropdown menu with '0-9' selected.
- Default Value:** A text input field containing the value '0'.
- Origin:** A dropdown menu with 'Reqs' selected.
- Buttons:** 'Save' and 'Cancel' buttons at the bottom.

Figure 63 Dependency Dialog

Recall from Chapter III Section A.3, that the “type” attribute currently does not provide any functionality in this version, but is provided for future extensions in which specific, pre-defined dependency type attributes can be associated at run-time with particular instantiated dependencies. Additionally, the “Value Range” is provided for future extensions and currently assumes a real value of 0 to 9.

The user can continue to create dependencies as needed. In fact, additional dependencies can be created at any time while using CASES.

5. Importing Data

Once the dependencies are created, the user will need to import data for each component and step versions. It is important to import all generic information first before importing specific dependency information. Generic information includes any non-dependency specific imported CSV file data (e.g. component ID and component name). Therefore, all dependencies will have the CSV file data as a common basis. Importing generic information overwrites all dependency information for a specific version where importing dependency information only affects a specific version and dependency values. The CSV files for the HelloWorld example are provided in the CASES CSV directory (note: the user will be required to create CSV files for their project schema). CSV files can contain as much dependency data as required or each

dependency data can be stored on separate files. A single file is a more efficient approach. Figure 64 shows how to use the popup menu to access the “Import CSV File” functionality for component Reqts version 1.1.

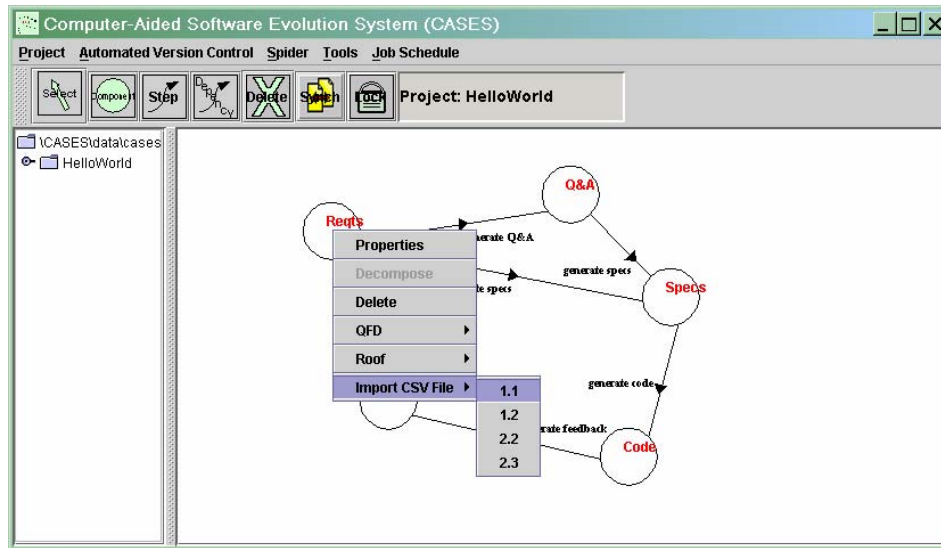


Figure 64 Import CSV Files

Once a version is selected, the confirmation dialog window will appear (see Figure 65). The user should press the Yes button for a dependency import and the No button for a generic import. As previously stated, it is important to import generic information prior to any dependency information; thus, the user should select “No” at this time.

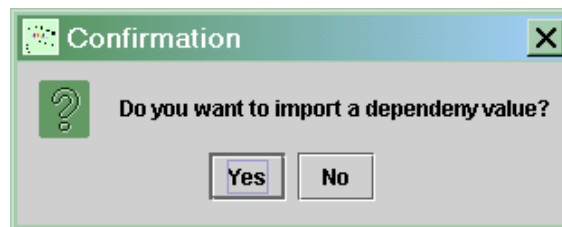


Figure 65 Import Dependency Values

Either selection will activate the functionality to open a CSV file for import, which requires the user to select a CSV file from the Open file dialog window (see Figure 66). As an example, Figure 66 illustrates the importing of the “HWRqtsvar1ver1.CSV” file (Hello World Requirements Variant 1, Version 1) into the “Rqts” component version 1.1.

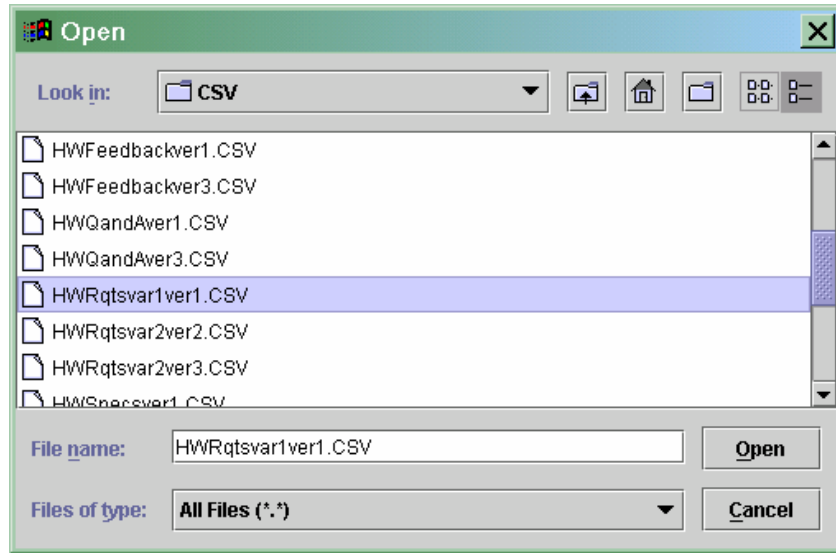


Figure 66 Open HelloWorld Reqts 1.1 CSV File

When the user returns later to import dependency value data, the Dependency List will be displayed (see Figure 67). The dependency list provides all dependencies created by the user. The user selects the dependency name for the component dependency data, which will be provided by data contained within the CSV file.

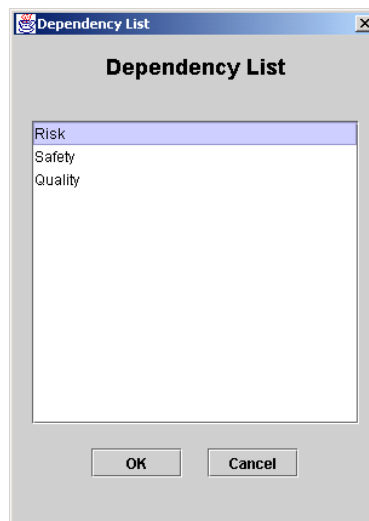


Figure 67 Dependency List

Once the user has selected a dependency, CASES will import column data into the component dependency JTable column. The user can specify which column to import by selecting it from the Column Header List (see Figure 68). In this example, the

dependency name and column header name is “Risk”, but matching names is left to the discretion of the user. It so happens that in this case, the portion of the CSV file the user is importing in Figure 68 has a column header also titled “Risk.”

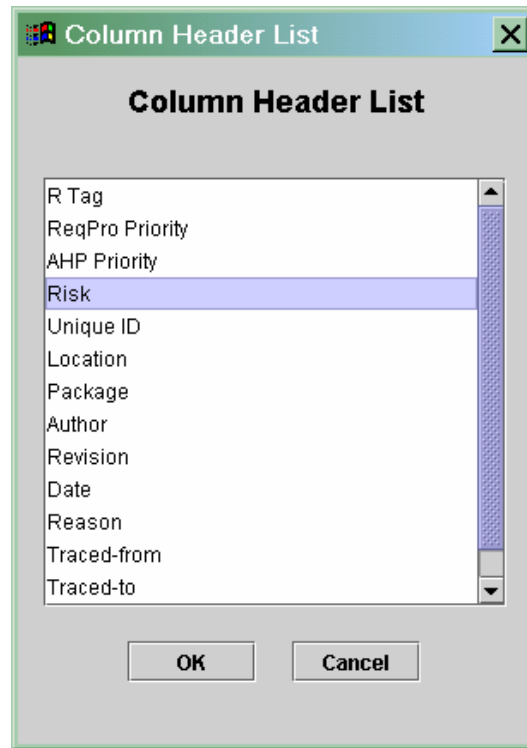


Figure 68 Column Header List

The user can continue this CSV importing process for all dependencies, steps, and components.

6. Deploying Dependencies

Recall from Chapter III Section B, that the user can perform individual calculations (upstream or downstream) at any time through the use of the “Calc” button on the HOQ window (see Figure 69). This will deploy the dependencies values within the single open QFD dialog. The user can perform collective calculations through the use of the “Sync” button on the CASES toolbar (see Figure 70). The “Sync” button will deploy dependency values upstream and downstream from the origin’s version throughout all QFD matrices (even those that are not open).

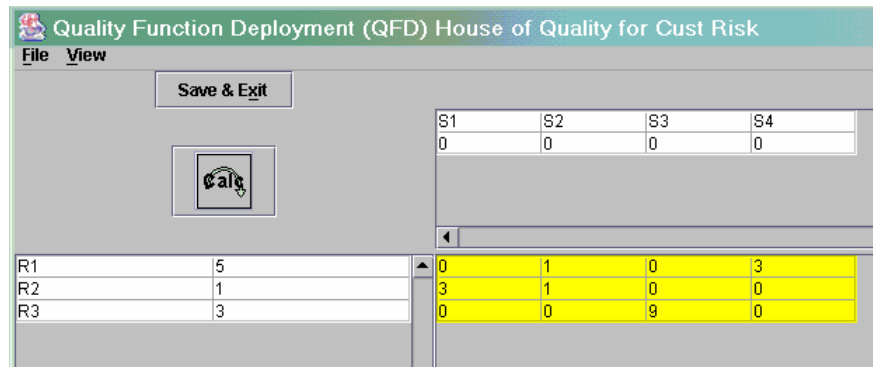


Figure 69 QFD Matrix for Risk Deployment Example

7. Viewing QFD Matrices

Once all data has been imported and dependencies values have been deployed, the user can observe the data by utilizing the popup menu QFD menu item (see Figure 70). This example illustrates how to view the Risk QFD matrix between “Rqts” and “Specs” version 1.1 artifacts.

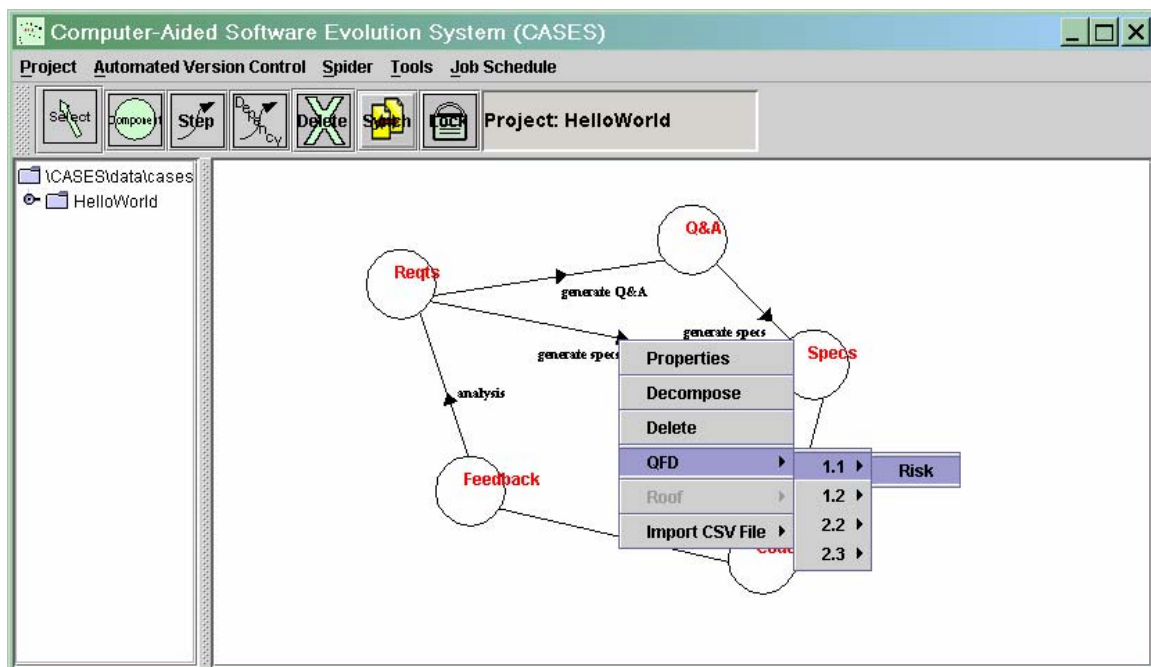


Figure 70 Open QFD Matrix Risk v.1.1

Figure 71 shows the synchronized values for the Risk QFD matrix between “Rqts” and “Specs” version 1.1. artifacts (note: the imported risk dependency values for “Rqts” is shown in the third column of the “Rqts” artifacts).

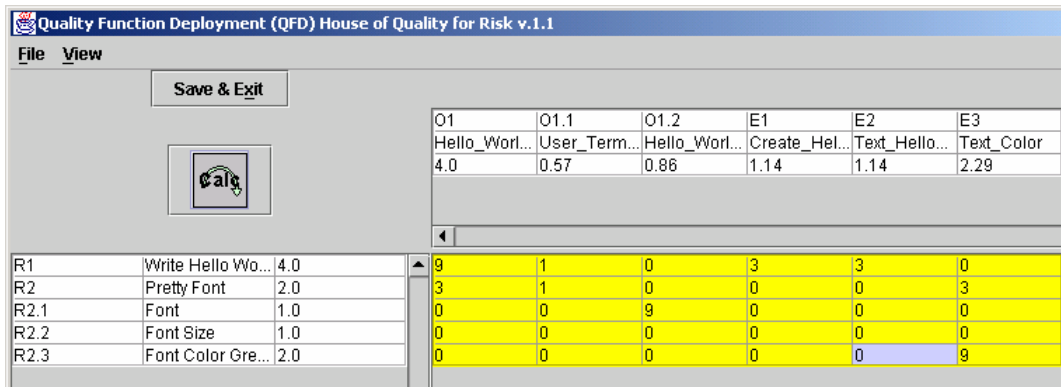


Figure 71 House of Quality Risk v.1.1

Recall from Chapter III Section C, that a user can store drivers and indicators in CASES. The user can access a Roof through the popup menu Roof menu item (see Figure 72). This example illustrates how to view the Risk Roof matrix between “Rqts” version 1.1 atomic components by right clicking on the “Rqts” component.

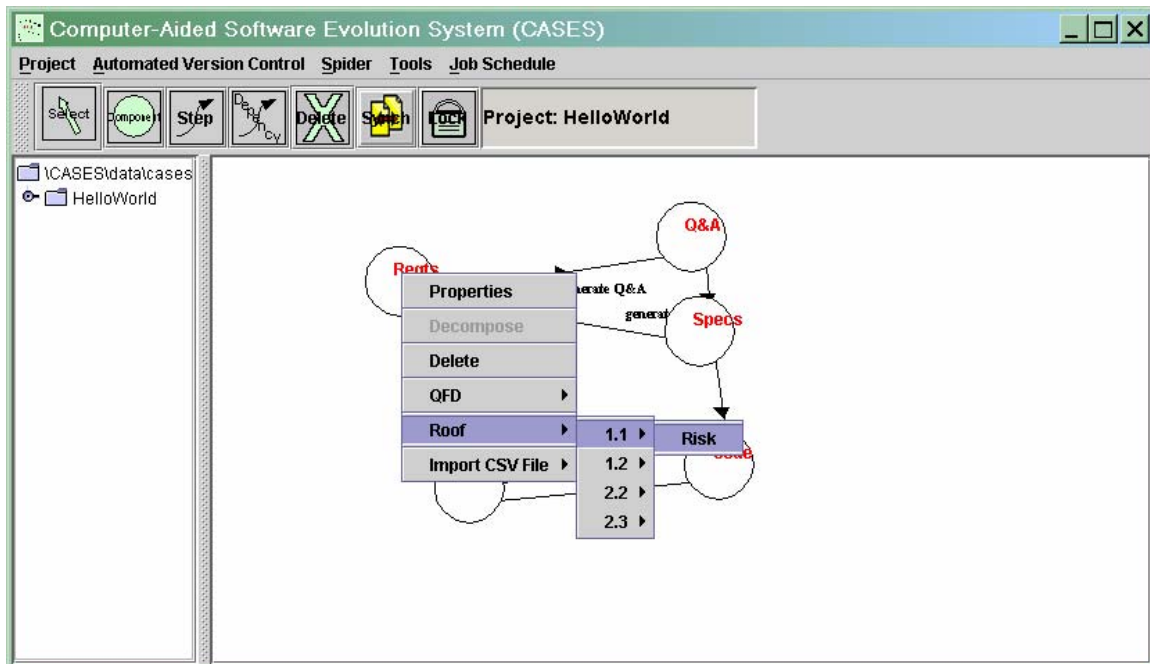


Figure 72 Roof Risk v1.1

CASES shows the atomic components in the House of Quality format with both component matrices identical. Drivers and indicator can be stored in the dependency

correlation matrix. In this example, Figure 73 show the indicator R1, the driver R2.3, and an independent atomic component R3.

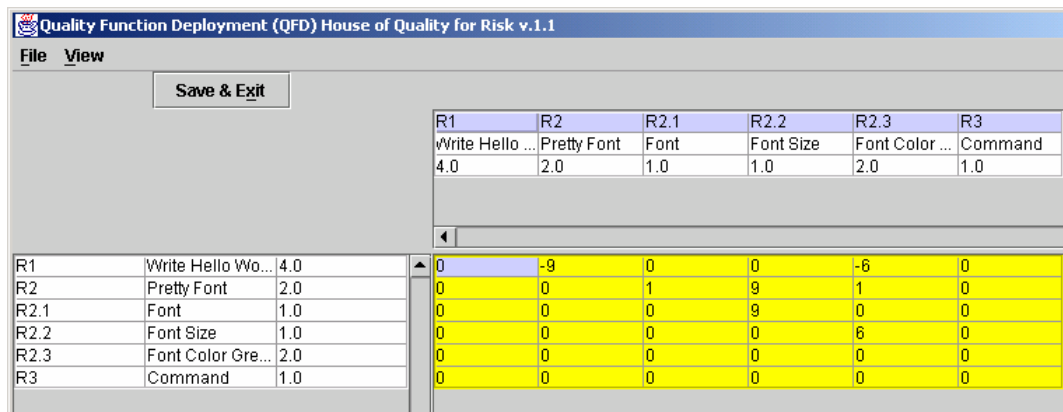


Figure 73 HOQ Roof

8. Engineering Views

Once all QFD dependency values have been deployed, the user can view specific subsets of dependency data by selecting the View menu from any House of Quality window. CASES 2.0 provides both a Dependency Threshold and a Component Trace view.

The Dependency Threshold view allows a user to isolate particular components that have dependency values greater than (or less than) a specified threshold value. The Dependency Threshold View Parameters dialog (see Figure 74) will be displayed when the user selects the dependency threshold menu item. The user must enter the 'n' value in the inequality for Equation 13. Example results are provided in Chapter III Section D.2.

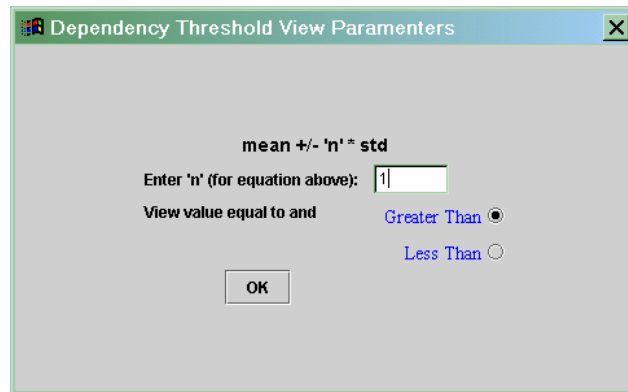


Figure 74 Dependency Threshold View

In the Component Trace View, the user identifies a single atomic component and then the tool isolates that portion of the underlying development effort associated with that component. The trace is performed by following the linkages from the designated component, through all connected components based on the strength of connection specified by the user. The Question dialog (see Figure 75) and then the Component Trace View dialog (see Figure 76) are displayed when the user selects the Component Trace menu item. The Question dialog allows the user to start a trace from a column or row of a particular QFD matrix.

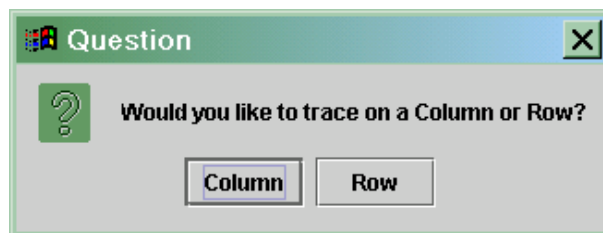


Figure 75 Column or Row Trace

In this example a row trace has been chosen, therefore available rows (e.g. R1 to R2.3) will be listed in the drop down list box in the Trace View dialog window. This trace example will show all artifacts, which are connected to R1 with a risk dependency value greater than or equal to 3. Example results are provided in Chapter III Section D.4.

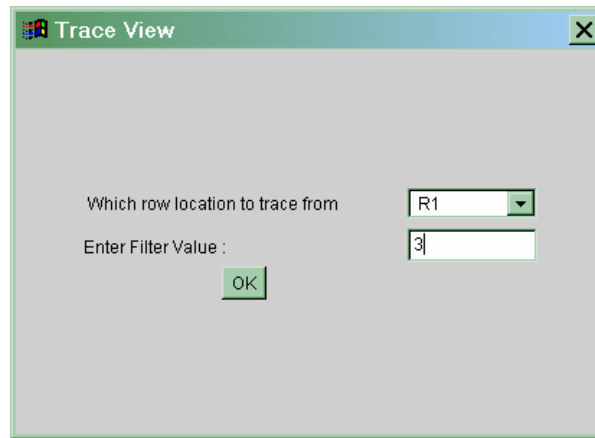


Figure 76 Trace View

9. SPIDER Functionality

SPIDER functionality can be accessed through the CASES Spider menu (see Figure 77). This functionality includes Edit, Decompose, Component Content, Step Content, and Trace. All SPIDER functionality has been modified from CASES 1.1 in CASES 2.0 to take advantage of an improved step identifier (this is an internal, derived requirement and is outside the scope of this user tutorial). The external functionality of SPIDER and Job Scheduler have not changed significantly from that presented in [LEHC99].

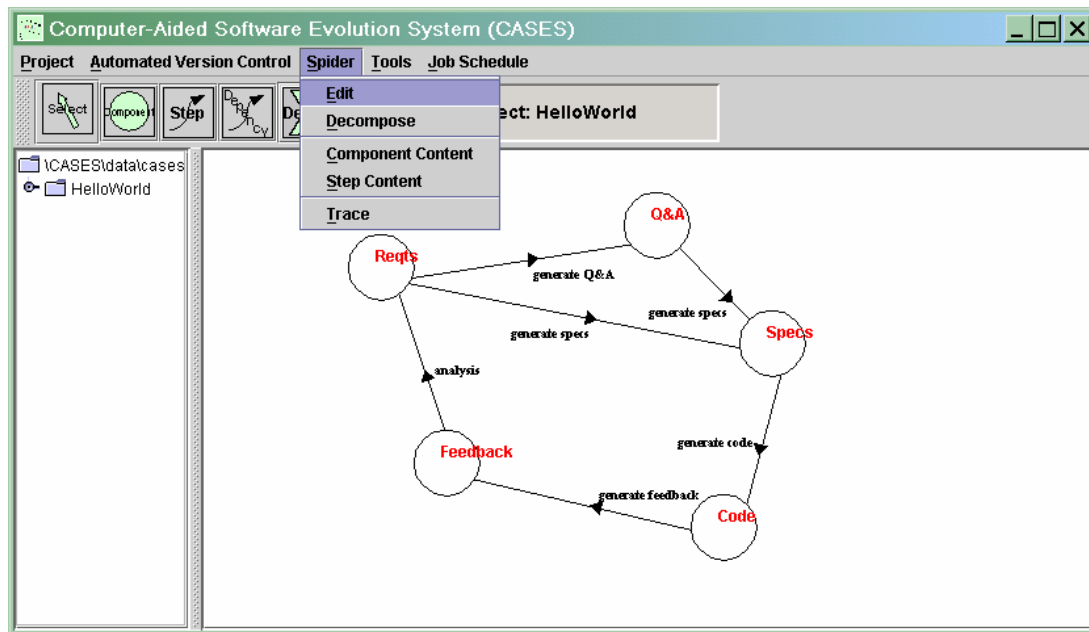


Figure 77 SPIDER

If a user selects the Edit, Decompose, or Component Content menu item from the Spider menu, then the Directory Tree dialog will be displayed (see Figure 78). Future versions should automate this step, because there is only one Component Content folder per opened step version.

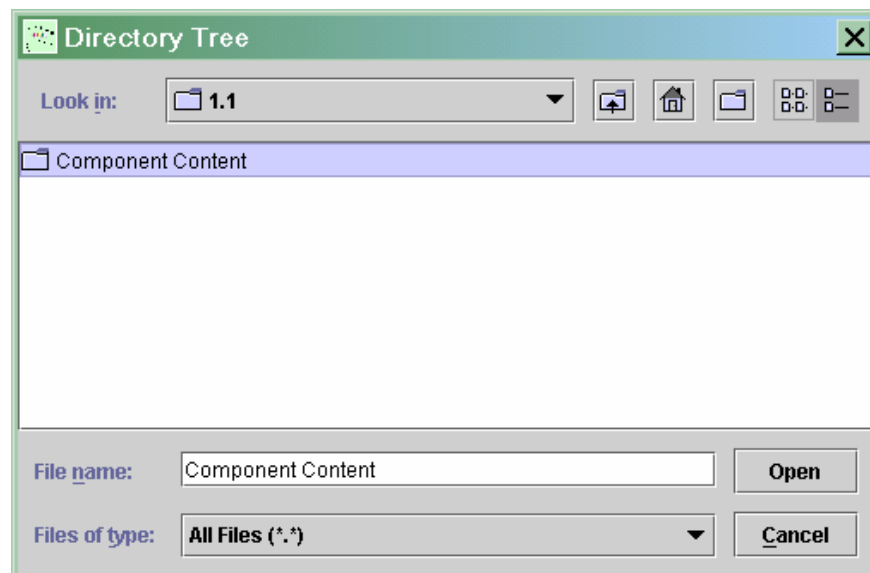


Figure 78 Directory Tree

After selecting the Component Content folder, the SPIDER Edit dialog will be presented to the user for selecting the Edit menu item. This window allows the user to change the primary and secondary input components for a step version.

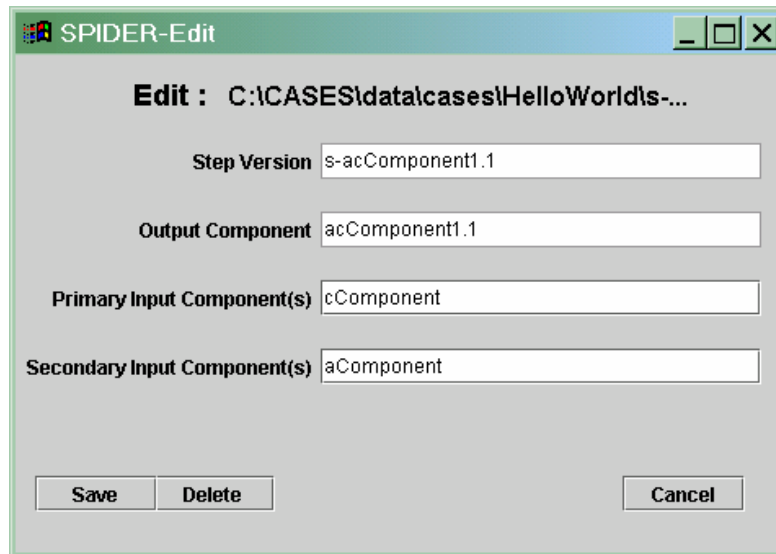


Figure 79 Edit SPIDER

Similar to the SPIDER Edit dialog, the Decompose dialog (see Figure 80) allows the user to view (but not edit) the decomposed components of a step version.

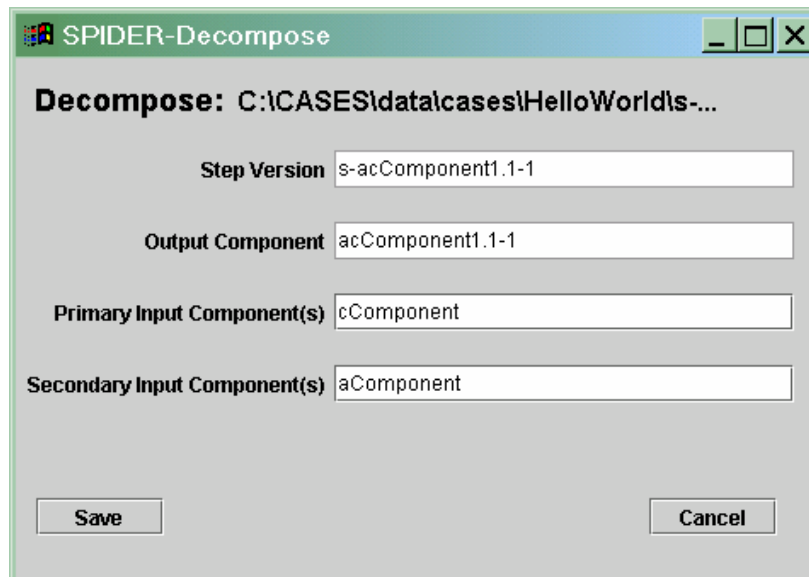


Figure 80 Decompose SPIDER

A user can add, edit, and delete link component files within the Component Content window (see Figure 81). These component file links range from text to ReqPro files.

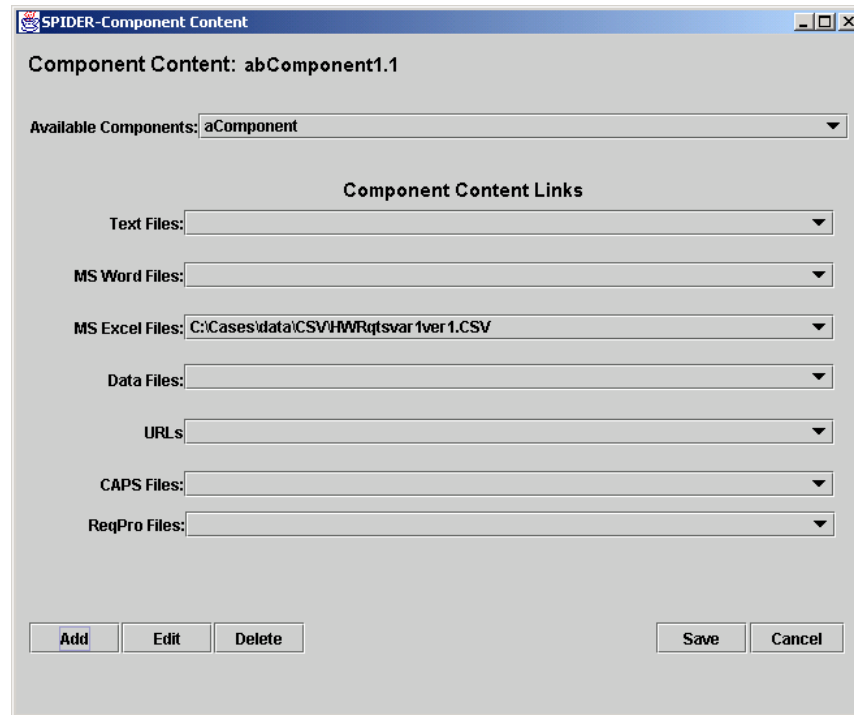


Figure 81 SPIDER Component Content

In this example, the “HWRqtsvar1ver1.CSV” file has been linked as an MS Excel file within the “Rqts” version 1.1 component (note: CASES 2.0 represents this component by the string “aComponent”). This can be accomplished by pressing the Add button, which will display the Component Content Editor window (see Figure 82).

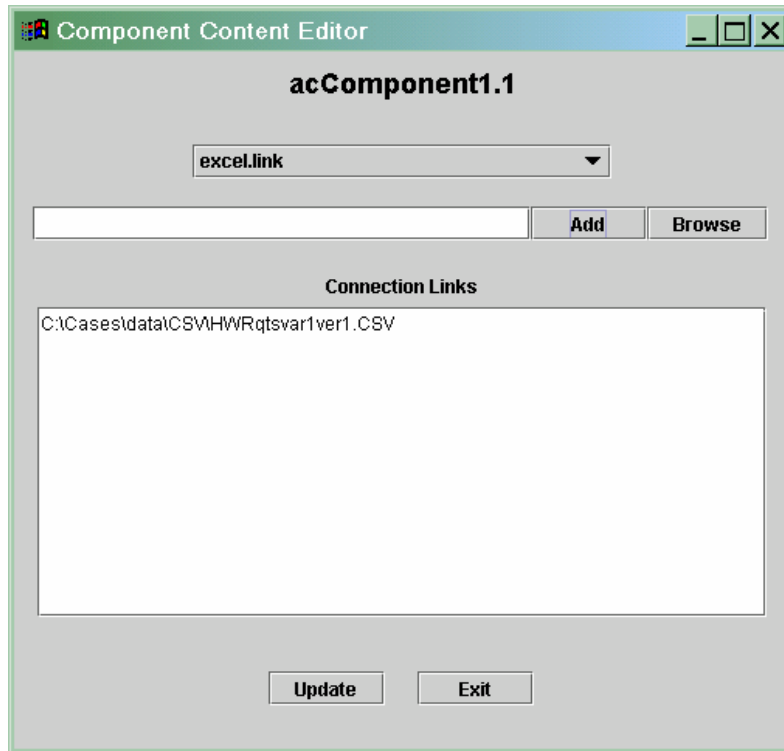
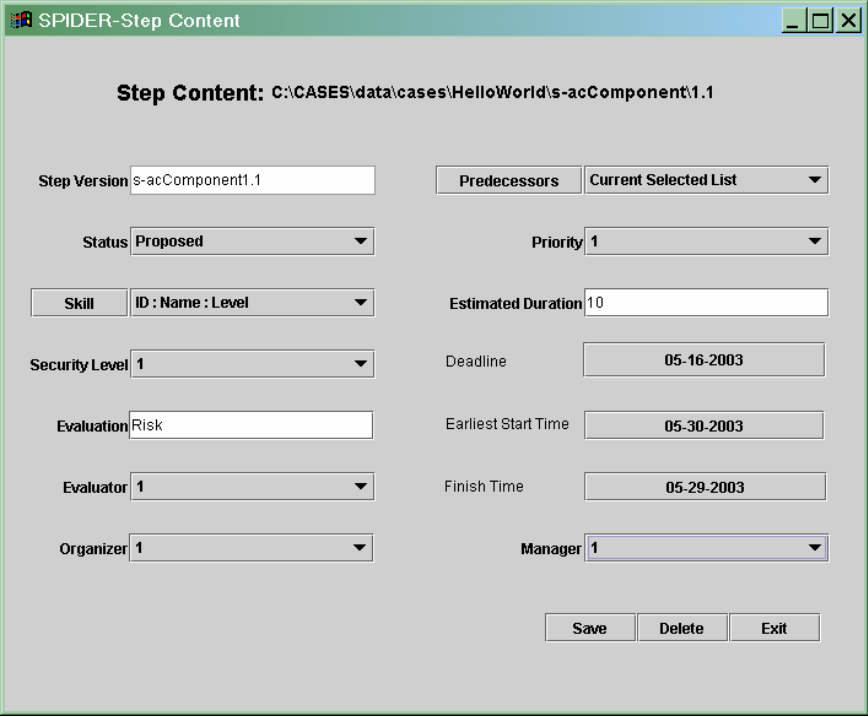


Figure 82 Component Content Editor

The user should first select the link type from the drop down list, browse for the required file, and then press the Add button to add it to the Connection Links window (note: several files may be added as needed). Once all files have been added, the Update button can be used to add file links to Component Content. The Exit button will close the Component Content Editor window.

As file links are assigned to component content, a user can assign activity information to step content (personnel, deadlines, skills, etc.) through the step content menu item in the Spider menu. Figure 83 shows the SPIDER Step Content dialog, which appears when the step content menu item is selected.



Step Content: C:\CASES\data\cases\HelloWorld\s-acComponent1.1

Step Version: s-acComponent1.1 Predecessors: Current Selected List

Status: Proposed Priority: 1

Skill: ID : Name : Level Estimated Duration: 10

Security Level: 1 Deadline: 05-16-2003

Evaluation: Risk Earliest Start Time: 05-30-2003

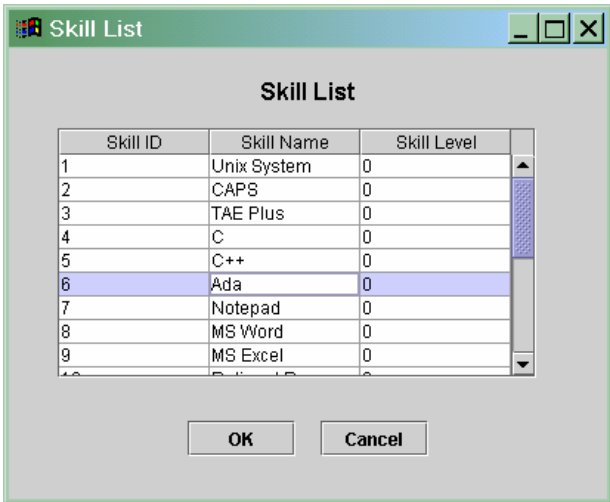
Evaluator: 1 Finish Time: 05-29-2003

Organizer: 1 Manager: 1

Save Delete Exit

Figure 83 SPIDER Step Content

The user can customize this window by editing the text boxes, selecting from the drop down lists, and by pressing defined buttons (Skill, Predecessors, Time or Deadline). If the user presses the Skill button, the Skill List dialog will appear (see Figure 84). A user can select specific information from this dialog (ID, Name, and Level).



Skill List

Skill ID	Skill Name	Skill Level
1	Unix System	0
2	CAPS	0
3	TAE Plus	0
4	C	0
5	C++	0
6	Ada	0
7	Notepad	0
8	MS Word	0
9	MS Excel	0

OK Cancel

Figure 84 Skill List

If the user presses the Predecessor button, the Predecessor List dialog will appear (see Figure 85). A user can then select the predecessor step. CASES 2.0 represents the steps with an identifier (e.g. “acComponent1.1” is the step from the ‘a’ Component to the ‘c’ Component version 1.1). CASES 1.1 had a less intuitive step identifier composed of characters only. Step identifiers require the user to recall initial identification of components prior to customization. Future improvements to CASES should remove this burden from the user.



Figure 85 Predecessor List

If the user presses the Deadline, Earliest Start Time, or Finish Time buttons, the Select Date dialog will appear (see Figure 86). CASES 1.1 was dependent upon the Visual Café calendar function (e.g. library `symantec.itools.awt.util.Calendar`) and Symantec events, but CASES 2.0 replaces that dependency with the standard Java library classes and events (see `CalendarDialog` Appendix B Section D).

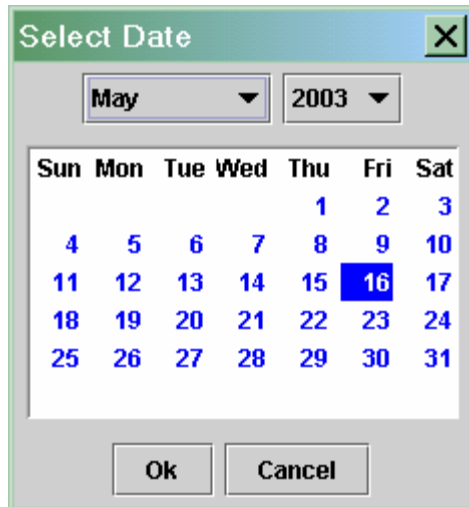


Figure 86 Calendar Dialog

Figure 87 illustrates the SPIDER Trace dialog, which appears when the user selects the Trace menu item in the CASES Spider menu.

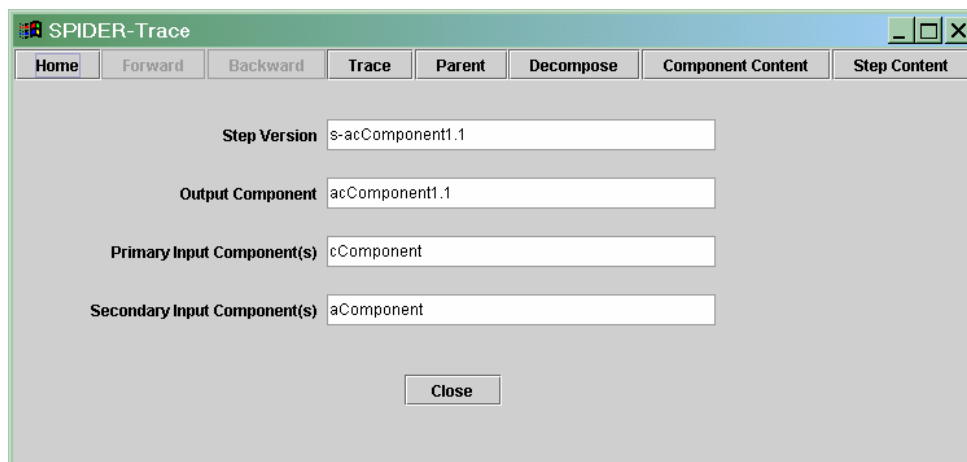


Figure 87 SPIDER Trace

10. Calling Tools

CASES provides a central location to call tools within the context of the Holistic Framework for Software Engineering as well as a personnel management tool (see Figure 88).

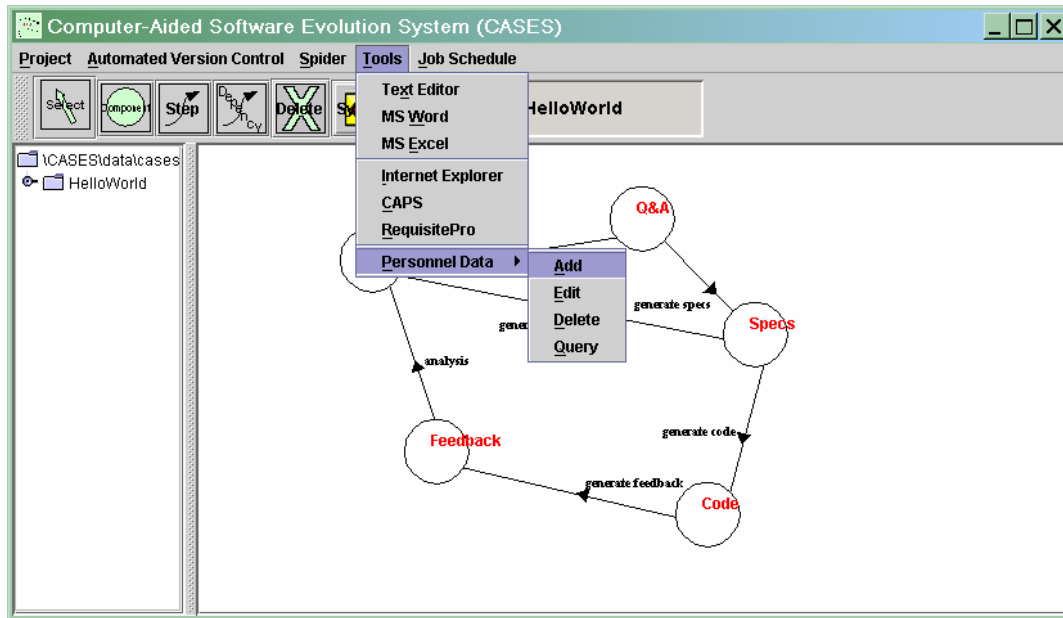


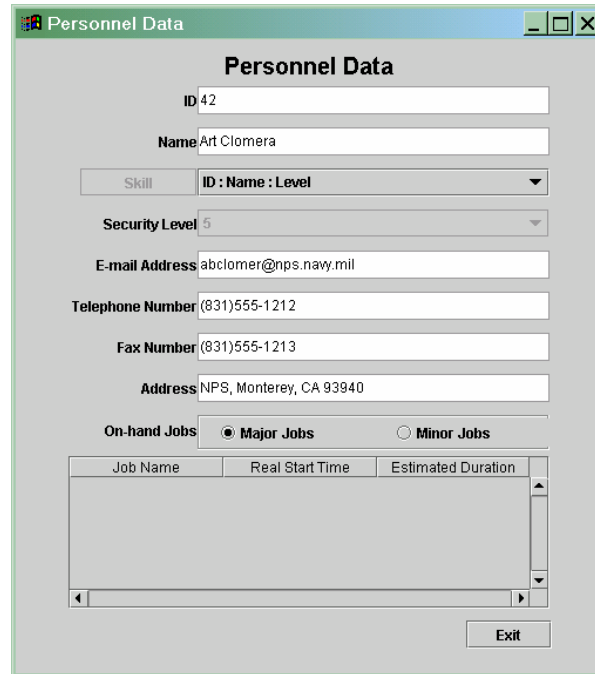
Figure 88 CASES Tools

11. Managing Personnel Data

Figure 89 shows the dialog window for adding personnel data (ID, Name, Skill, Security Level, email, telephone, fax, and address). The personnel data is stored in the stakeholder directory under the ID number; therefore, a user should ensure that ID numbers are unique (note: pressing the Skill button will allow the user to modify the skill information within the personnel data).

Figure 89 Add Personnel Data

Editing is similar to adding personnel data, but includes major and minor job information (see Figure 90).

A Windows-style dialog box titled "Personnel Data". It contains several input fields: "ID" with the value "42", "Name" with "Art Clomera", "Skill" with a dropdown menu showing "ID : Name : Level", "Security Level" with a dropdown menu showing "5", "E-mail Address" with "abclomer@nps.navy.mil", "Telephone Number" with "(831)555-1212", "Fax Number" with "(831)555-1213", and "Address" with "NPS, Monterey, CA 93940". Below these fields are two radio buttons for "On-hand Jobs": "Major Jobs" (selected) and "Minor Jobs". At the bottom is a table with three columns: "Job Name", "Real Start Time", and "Estimated Duration". The table is currently empty. An "Exit" button is located at the bottom right of the dialog box.

Job Name	Real Start Time	Estimated Duration

Figure 90 Edit Personnel Data

12. Job Scheduling

The Job Schedule menu provides the user access to scheduling and job assignment. CASES 2.0 provides the same functionality developed in CASES 1.1 which only includes displaying an Error warning (see Figure 91).

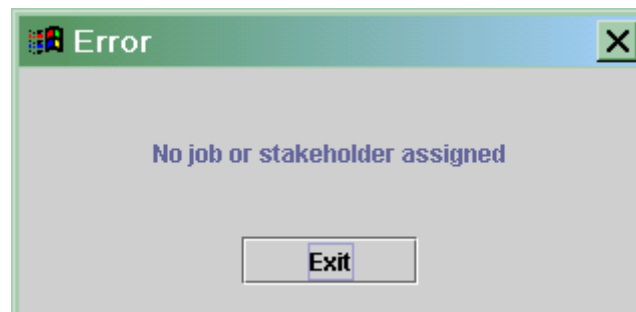


Figure 91 No Job or Stakeholder Assigned Error

13. Deleting a Project

Finally, after a project is no longer needed – it can be deleted. To delete a project, a user selects the Delete Project menu item from the CASES Project menu. Figure 92 illustrates deleting the HelloWorld project.

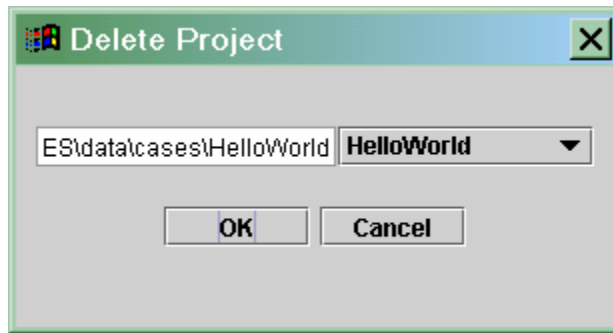


Figure 92 Delete Project

APPENDIX B. CASES SOURCE CODE

A. CASES PACKAGE

1. Cases.CasesFrame

```
/**-----
 * Filename: CasesFrame.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1 and removed Visual Cafe' dependencies
 * Description: main frame of Cases. It is the main frame to connect to all
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_Cases
 * Modifications include a tool bar and draw pane.
 *-----
 */
package Cases;

import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.util.StringTokenizer;
import java.util.Vector;
import javax.swing.*;
import Cases.GUI.*;
import Cases.GUI.avc.AVCMenu;
import Cases.Interfaces.I_Cases;
import Cases.QFD.util.ObjectFileOperations;

/** CasesFrame : main frame of Cases. It is the main frame to connect to all
 * frames in Cases system.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_Cases
 */
public class CasesFrame extends JFrame implements CasesTitle, I_Cases
{
    /**
     * The width of the frame.
     */
    private final int INITIAL_WIDTH = 800;

    /**
     * The height of the frame.
     */
    private final int INITIAL_HEIGHT = 600;

    /**
     * fileChooser : instantiate new JFileChooser with the current directory is Cases
     */
    public JFileChooser fileChooser = new JFileChooser(CASESDIRECTORY);

    /**
     * projectAtomicVector : a vector contains all atomics in the project.
     * It is used in Job Schedule.
     */
    public Vector projectAtomicVector = new Vector();

    /**
     * projectStepContentVector : a vector contains all StepContent objects in the project
     * It is used in Job Schedule
     */
}
```

```

    */
    public Vector projectStepContentVector = new Vector();

    /**
     * stepContentPathVector : a vector contains the paths of all step contents in the project
     * It is used in Job Schedule
     */
    public Vector stepContentPathVector = new Vector();

    /**
     * personnelVector : a vector contains all Personnel objects in the project
     * It is used in Job Schedule
     */
    public Vector personnelVector = new Vector();

    /**
     * scheduleAtomicVector : a vector contains all StepContent objects whose
     * status are "Scheduled"
     * It is used in Job Schedule
     */
    public Vector scheduledAtomicVector = new Vector();

    /**
     * vc : a VersionControl object
     */
    public VersionControl vc = null;

    /**
     * title : a title of the current menu item under SPIDER menu,
     * e.g. Edit, Decompose, Component Content, Step Content, or Trace
     */
    public String title = null;

    /**
     * projectName : a current project name, e.g. C4I, C3I, ...
     */
    public String projectName = null;

    /**
     * pathName : the absolute path of the current project, e.g. C:\Cases\C4I
     */
    public String pathName = CASESDIRECTORY.getAbsolutePath();

    ///Job Schedule Variables
    public Vector person_queue;
    public Vector job_queue = new Vector();
    public Vector job_queue1 = new Vector();
    public Vector job_queue2 = new Vector();
    public Vector job_pool = new Vector();
    public int personid=0;
    public int ctrl=0;

    public CasesToolBar drawToolBar;
    public GraphPanel drawPanel;
    public ProjectSchemaFrame psfWindow;
    public Dialog dialog;
        boolean frameSizeAdjusted = true;
        JMenuBar casesMenuBar = new JMenuBar();
        ProjectMenu projectMenu = new ProjectMenu(this);
    AVCMenue avcMenu = new AVCMenue(this);
        ToolsMenu toolsMenu = new ToolsMenu(this);
        SpiderMenu spiderMenu = new SpiderMenu(this);

        JobScheduleMenu jobScheduleMenu = new JobScheduleMenu(this);

    public JTabbedPane jTabbedPane1;
    public FileSystemModel fileSystemModel;
    public JTree fileTree;

```

```

/**
 * Create CasesFrame
 */
public CasesFrame() {
    fileSystemModel = new FileSystemModel (CASESDIRECTORY);
    fileTree = new JTree (fileSystemModel);
    fileTree.setEditable(false);
    dialog = new Dialog(this,true);
    initGUI();
}

/** main procedure for initializing the GUI */
private void initGUI() {
    this.setMenuBar(casesMenuBar);
    this.setTitle("Computer-Aided Software Evolution System (CASES)");
    getContentPane().setLayout(new java.awt.BorderLayout());
    getContentPane().setForeground(java.awt.Color.blue);
    setSize (INITIAL_WIDTH, INITIAL_HEIGHT);
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    setLocation((screenSize.width - INITIAL_WIDTH) / 2,
(screenSize.height - INITIAL_HEIGHT) / 2);
    setVisible(false);
    setBounds(new java.awt.Rectangle(0,0,831,479));
    setIconImage((new javax.swing.ImageIcon("/CASES/IMAGES/Cases.jpg")).getImage());
    // construct the Cases menu
    casesMenuBar.add(projectMenu);
    casesMenuBar.add(avcMenu);
casesMenuBar.add(spiderMenu);
    casesMenuBar.add(toolsMenu);
    casesMenuBar.add(jobScheduleMenu);

    /**
     * Check and Create Cases and stakeholder directory
     */
    if( CASESDIRECTORY.isDirectory() ){
        String[] fileList = CASESDIRECTORY.list();
        if( fileList.length > 0 ){
            setOpenDelete( true );
        }
        else if( fileList.length == 0 ){
            setOpenDelete( false );
        }
    }
    else{
        CASESDIRECTORY.mkdir();
        setOpenDelete( false );
    }

    if( !STAKEHOLDER.isDirectory() ){
        STAKEHOLDER.mkdir();
    }
    menuSetEnabled(false);

    // variable for cases toolbar
    drawToolBar = new CasesToolBar(this);
    // variable for draw pane
    drawPanel = new GraphPanel (this);
    // a split pane to separate file directory tree and draw pane
    JSplitPane splitPane = new JSplitPane(
// make it a horizontal split
JSplitPane.HORIZONTAL_SPLIT, true,
// left side is the file tree
new JScrollPane( fileTree ),
// right side is the draw pane
new JScrollPane( drawPanel ) );

```

```

getContentPane().add(splitPane, BorderLayout.CENTER );
        getContentPane().setBackground(Color.white);
        getContentPane().add(drawToolBar, BorderLayout.NORTH);
// customize the file tree
fileTree.setBorder(BorderFactory.createEtchedBorder());
fileTree.putClientProperty("JTree.lineStyle", "Horizontal");
        fileTree.putClientProperty("JTree.lineStyle", "Angled");
    }

    /**
    * method to control the visibility based upon b
    * @param boolean
    */
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

    /**
    * a main procedure for unit testing
    */
    static public void main(String[] args) {
        CasesFrame cf = new CasesFrame();
        cf.addWindowListener( new WindowAdapter() {
            public void windowClosing( WindowEvent e ) {
                System.exit(0);
            }
        });
        cf.setVisible(true);
    }

    /**
    * openDependency for drawFrame and save dependency data
    * to the depAttrib.cfg file.
    * The dependencies are constrained by MAXDEPENDENCIES global variable
    * defined in CasesTitle.java.
    */
    public void openDependencyFrame () {
        if (drawPanel.depAttrib.size() < MAXDEPENDENCIES) {
            DepAttributes depAttribute = new DepAttributes(dialog,this);
            depAttribute.show();
            if (depAttribute.cancelFlag == false) {
                DependencyType dt = new DependencyType();
                dt.setName(depAttribute.getShortName());
                dt.setDescription(depAttribute.getDescription());
                dt.setType(depAttribute.getTypeSelected());
                dt.setValueRange(depAttribute.getRangeSelected());
                dt.setDefaultValue(depAttribute.getDefaultValue());
                dt.setOrigin(depAttribute.getDepOrigin());
                dt.setDepVersion(depAttribute.getDepVersion());
                drawPanel.depAttrib.add(dt);
                depAttribute.dispose();
                ObjectFileOperations ofo = new ObjectFileOperations();
                ofo.fileSaveActionPerformed(pathName+"\\depAttrib.cfg",drawPanel.depAttrib);
                this.drawToolBar.setCalcButton(true);
            }
        } else {
            JOptionPane.showMessageDialog(null, "Max Dependencies Reached", "Limit Reached",
            JOptionPane.INFORMATION_MESSAGE);
        }
    }

    /**
    * overrides the super addNotify() method.
    */
    public void addNotify()
    {

```

```

        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        JMenuBar menuBar = getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

    /**
     * Short cut to print the output
     * @param string : the output string
     */
    public void debug( String string ){
        System.out.println(string);
    }

    /**
     * Gray out Open Project and Delete Project menu items
     * if there is no project under CASESDIRECTORY .
     * Or not gray out if CASESDIRECTORY exists at least one project
     */
    public void setOpenDelete( boolean enabled ){
        this.projectMenu.openProjectMenuItem.setEnabled(enabled);
        this.projectMenu.deleteProjectMenuItem.setEnabled( enabled );
    }

    /**
     * If this.projectName is null, flag = false
     * Else flag = true
     */
    void menuSetEnabled( boolean flag){
        avcMenu.setEnabled(flag);

        this.spiderMenu.setEnabled( flag );
        this.toolsMenu.setEnabled( flag );
        this.jobScheduleMenu.setEnabled(flag);
    }

    /**
     * Get VersionControl object from current.vsn file
     */
    public void getVersionControl(){
        try{
            FileInputStream fileInput = new FileInputStream(this.pathName+"\\current.vsn");
            ObjectInputStream current = new ObjectInputStream( fileInput );
            if( current != null ){
                this.vc = (VersionControl) current.readObject();
            }
            current.close();
            fileInput.close();
        }
        catch( ClassNotFoundException e ){ debug("ClassNotFoundException: "+e); }
        catch(IOException i){
            JOptionPane.showMessageDialog(this, "Can not open the application \nsince current.vsn does not exist in this project.",
                "Error Message", JOptionPane.ERROR_MESSAGE);
        }
        return;
    }
}

```

```

        if( this.vc != null ){
            directoryTree();
        }
    }

/**
 * Check and locate the current step in fileChooser
 * If the selected step is not a current step, Error Message shows up
 */
    public void directoryTree(){
        String currentStep = (String)this.vc.getCurrentStep();
        String currentVersion = (String)this.vc.getCurrentVersion();
        String wholePath = this.pathName+ "\\ "+currentStep+"\\ "+currentVersion;
        File file = new File(wholePath);
        int returnValue=-1;
        this.fileChooser = new JFileChooser(file);
        this.fileChooser.setDialogTitle("Directory Tree");
        this.fileChooser.setCurrentDirectory(file);
        this.fileChooser.setSelectionMode(this.fileChooser.DIRECTORIES_ONLY);
        returnValue = this.fileChooser.showDialog(this,"Open");

        if( returnValue == this.fileChooser.APPROVE_OPTION ){
            File aFile = this.fileChooser.getCurrentDirectory();
            String absPath = aFile.getAbsolutePath();
            String output = currentStep.substring((int)currentStep.indexOf("-")+1)+currentVersion;
            if( absPath.equals( wholePath ) ){
                debug("CasesFrame:currentDir:absPath:"+absPath);
                currentDir(absPath);
            }
            else if( absPath.length() > wholePath.length() ){
                debug("CasesFrame:subDir:absPath:"+absPath);
                subDir(absPath);
            }
            else {
                JOptionPane.showMessageDialog(this, "The selected directory is not the current step.",
                    "Error Message", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

/**
 * Locate the selected step and it is an original step
 *
 * @param absPath : a string of the whole path
 */
    public void currentDir(String absPath ){
        String currentStep = (String)this.vc.getCurrentStep();
        String currentVersion = (String)this.vc.getCurrentVersion();
        String stepName = currentStep + currentVersion;
        String output = currentStep.substring((int)currentStep.indexOf("-")+1)+currentVersion;
        // debug statements to upgrade old version to Cases 2.0
        debug("CasesFrame:currentDir:currentStep:"+currentStep);
        debug("CasesFrame:currentDir:currentVersion:"+currentVersion);
        debug("CasesFrame:currentDir:stepName:"+stepName);
        debug("CasesFrame:currentDir:output:"+output);

        if( this.title.equals(EDIT_TITLE) ){
            (new EditDecomposeFrame(this.title,stepName, output,absPath)).setVisible(true);
        }
        else if( this.title.equals(DECOMPOSE_TITLE) ){
            int theMax = (int)findMax(absPath) +1 ;
            stepName = stepName + "-" + theMax;
            String decomposeOutput = stepName.substring(2);
            (new EditDecomposeFrame(this.title,stepName,
                decomposeOutput,absPath,theMax)).setVisible(true);
        }
        else if( this.title.equals(COMPONENT_CONTENT_TITLE) ){

```



```

        setComponentContent( stepName, absPath);
    }
    else if( this.title.equals(STEP_CONTENT_TITLE) ){
        setStepContent(null, stepName, absPath);
    }
    else if( this.title.equals	TRACE_TITLE ) ){
        setTraceFrame(stepName, absPath);
    }
}

/**
 * Locate the selected step and it is a decompose step
 *
 * @param absPath : a string of the whole path
 */
public void subDir(String absPath ){
    String currentStep = (String)this.vc.getCurrentStep();
    String currentVersion = (String)this.vc.getCurrentVersion();
    String wholePath = this.pathName + "\\\"+currentStep+"\\\"+currentVersion;
    String newSubString = absPath.substring(0, wholePath.length());
    String output = currentStep.substring((int)currentStep.indexOf("-")+1)+currentVersion;
    // debug statements to upgrade old version to Cases 2.0
    debug("CasesFrame:subDir:currentStep:"+currentStep);
    debug("CasesFrame:subDir:currentVersion:"+currentVersion);
    debug("CasesFrame:subDir:wholePath:"+wholePath);
    debug("CasesFrame:subDir:newSubString:"+newSubString);
    debug("CasesFrame:subDir:output:"+output);

    int result = wholePath.compareTo(newSubString);
    if( result == 0 ){
        String subName = absPath.substring(wholePath.length()+1);
        StringTokenizer st = new StringTokenizer(subName,"\\");
        Vector stVector = new Vector();
        while( st.hasMoreTokens()){
            stVector.addElement(st.nextToken());
        }
        String stResult = (String)stVector.elementAt(0);
        for( int i=1; i<stVector.size(); i++){
            stResult = stResult+"."+ (String)stVector.elementAt(i);
        }
        output = output + "-" + stResult;
        String stepName = currentStep+currentVersion+"-"+stResult;

        if( this.title.equals(EDIT_TITLE) ){
            (new EditDecomposeFrame(this.title, stepName, output, absPath)).setVisible(true);
        }

        else if( this.title.equals(DECOMPOSE_TITLE) ){
            int theMax = (int)findMax(absPath) + 1;
            stepName = stepName + "." + theMax;
            String decomposeOutput = stepName.substring(2);
            (new EditDecomposeFrame(this.title, stepName, decomposeOutput, absPath, theMax)).setVisible(true);
        }

        else if( this.title.equals(COMPONENT_CONTENT_TITLE) ){
            setComponentContent( stepName, absPath);
        }
        else if( this.title.equals(STEP_CONTENT_TITLE) ){
            setStepContent(null, stepName, absPath);
        }
        else if( this.title.equals	TRACE_TITLE ) ){
            setTraceFrame(stepName, absPath);
        }
    }
    else{
        JOptionPane.showMessageDialog(this, "The selected directory is not the current step.",
            "Error Message", JOptionPane.ERROR_MESSAGE);
    }
}

```

```

    }

/**
 * Return a vector of all components of the current step
 *
 * @param stepName : the selected step
 * @param absPath : the complete path of this stepName
 * @return componentVector : a vector of strings holds component names
 */
public Vector getComponents(String stepName, String absPath){
    Vector componentVector = new Vector();
    String s = stepName.substring(2);
    componentVector.addElement(s);
    try{
        FileInputStream fileInput = new FileInputStream(absPath+"\\input.p");
        DataInputStream inputP = new DataInputStream(fileInput);
        if( inputP != null ){
            String sP = inputP.readLine();
            if( ( sP != null ) && (!sP.equals("")) ){
                tokenizer(componentVector, sP);
            }
        }
        inputP.close();
        fileInput.close();

        fileInput = new FileInputStream(absPath+"\\input.s");
        DataInputStream inputS = new DataInputStream(fileInput);
        if( inputS != null ){
            String sS = inputS.readLine();
            if( ( sS != null ) && (!sS.equals("")) ){
                tokenizer(componentVector, sS);
            }
        }
    }
    catch( IOException e ){
        debug("IOException at ComponentContent: "+e);
    }
    return componentVector;
}

/**
 * Connect to ComponentContentFrame
 *
 * @param stepName : selected step name, e.g., s-I, 1.1, 1, ...
 * @param absPath : the absolute path of stepName, e.g., F:\Cases\s-I\1.1
 */
public void setComponentContent( String stepName, String absPath ){

    (new ComponentContentFrame(this.projectName, stepName.substring(2), getComponents(stepName, absPath),
    (Vector)fileNameList()).setVisible(true);
}

/**
 * Connect to StepContentFrame
 *
 * @param traceFrame : instantiation of TraceFrame
 * @param stepName : selected step name, e.g., s-I, 1.1, 1, ...
 * @param absPath : the absolute path of stepName, e.g., F:\Cases\s-I\1.1
 */
public void setStepContent(TraceFrame traceFrame, String stepName, String absPath ){
    File file = new File(absPath);
    debug("CasesFrame:file.getName():");System.out.println(file.getName());
    debug("CasesFrame:file.isDirectory():");System.out.println(file.isDirectory());
    if( file.isDirectory() ){
        if( !(file.getName().equals(COMPONENT_CONTENT_DIR)) ){
            Vector atomicsVector = new Vector();
            File temp = new File(this.pathName, this.vc.getCurrentStep());
            String[] theFiles = temp.list();

```

```

        for( int i=0; i<theFiles.length; i++){
            getAtomsics(new File(temp, theFiles[i]), atomsicsVector);
        }
        (new StepContentFrame(this, traceFrame, stepName, absPath,
                               atomsicsVector)).setVisible(true);
    }
    else{
        JOptionPane.showMessageDialog(this, absPath+" is not a step. Please reselect it!",
                                       "Warning Message",JOptionPane.ERROR_MESSAGE);
        return;
    }
}
else{
    JOptionPane.showMessageDialog(this, absPath+" is not a step. Please reselect it!",
                                   "Warning Message",JOptionPane.ERROR_MESSAGE);
    return;
}
}
}

/**
 * Connect to TraceFrame
 *
 * @param stepName : selected step name, e.g., s-I, 1.1, 1, ...
 * @param absPath : the absolute path of stepName, e.g., F:\Cases\s-I\1.1
 */
public void setTraceFrame(String stepName, String absPath){
    (new TraceFrame(this, this.pathName, getComponents(stepName, absPath), (Vector)fileNameList()).setVisible(true);
}

/**
 * Get all atomsics of the selected step
 *
 * @param aFile : since a file is a step, aFile is a selected step
 * @param storedVector : a vector of atomsics
 */
public void getAtomsics(File aFile, Vector storedVector){

    boolean isAtomic = true;
    if( (aFile.isDirectory()) && !((aFile.getName()).equals(COMONENT_CONTENT_DIR)) ){
        String[] theFiles = aFile.list();
        for( int i=0; i<theFiles.length; i++){
            char c = (char)((String)theFiles[i]).charAt(0);
            Character theChar = new Character(c);
            if(theChar.isDigit(c)){
                File f1 = new File( aFile, theFiles[i]);
                if( (f1.isDirectory()) && !((f1.getName()).equals(COMONENT_CONTENT_DIR)) ){
                    isAtomic = false;
                    getAtomsics(new File(aFile, theFiles[i]), storedVector );
                }
            }
        }
        if( isAtomic ){
            storedVector.addElement(aFile);
        }
    }
}

/**
 * Find a maximum number of the selected step version number
 *
 * @return theMax : an integer, the maximum number of the selected step
 * @param thePath : the path of selected step, e.g., F:\Cases\s-I\1.1\2\1
 */
public int findMax( String thePath ){
    Vector v = new Vector();
    File f1 = new File( thePath );
    String[] list = (String[]) f1.list();
    for(int i=0; i<list.length;i++){

```

```

        File f2 = new File(thePath, list[i]);
        if( f2.isDirectory() ){
            try{
                Integer i1 = new Integer(list[i]);
                v.addElement(i1);
            }
            catch(NumberFormatException n){
            }
        }
    }

    int theMax = 0;
    for(int i=0; i<v.size(); i++){
        theMax = Math.max(theMax,(int)((Integer)v.elementAt(i)).intValue());
    }
    return theMax;
}

/**
 * Token a string with delimit is " , "
 *
 * @param v : a vector of tokenizer string
 * @param s : a tokenized string
 */
public void tokenizer( Vector v, String s){
    StringTokenizer st = new StringTokenizer(s," , ");
    while(st.hasMoreTokens()){
        String theString = (st.nextToken()).trim();
        v.addElement(theString);
    }
}

/**
 * List all steps of a project
 * @return a vector of string with all steps
 */
public Vector fileNameList(){
    Vector v = new Vector();
    String[] theList = (String[])(new File(this.pathName)).list();
    for( int i=0; i<theList.length;i++){
        if( new File(this.pathName+"\\ "+theList[i]).isDirectory()){
            if( (((String)theList[i]).charAt(1) == '\\') { // filter out any garbage in directory
                v.addElement(theList[i]);
            }
            debug("CasesFrame:fileNameList:theList[i]: "+theList[i]);
        }
    }
}

return v;
}

/**
 * Save all personnel object after updating
 */
public void savePersonnelVector(Vector v){
    personnelVector = v;
    if( personnelVector != null ){
        for( int i=0; i<personnelVector.size(); i++ ){
            Personnel personnel = (Personnel)personnelVector.elementAt(i);
            if( personnel != null ){
                try{
                    FileOutputStream fileOutput = new FileOutputStream(STAKEHOLDER+"\\ "+personnel.getID());
                    ObjectOutputStream oo = new ObjectOutputStream(fileOutput);
                    if( oo != null ){
                        oo.writeObject(personnel);
                    }
                    oo.flush();
                    oo.close();
                    fileOutput.close();
                }
            }
        }
    }
}

```

```

        catch(IOException io){
            JOptionPane.showMessageDialog(this, "Sorry, saving is unsuccessful", "Error Message",
JOptionPane.ERROR_MESSAGE);
        }
    }
}

/**
 * Return a vector of personnel objects under stakeholder directory
 */
public Vector getPersonnelVector(){
    personnelVector = new Vector();
    String[] list = (String[])STAKEHOLDER.list();
    if( list.length > 0 ){
        for( int i=0; i<list.length; i++ ){
            try{
                String filePath = STAKEHOLDER.getAbsolutePath()+list[i];
                File aFile = new File(filePath);
                if( aFile.exists() ){
                    FileInputStream fileInput = new FileInputStream(aFile);
                    ObjectInputStream oi = new ObjectInputStream(fileInput);
                    if( oi != null ){
                        Personnel personnel = (Personnel)oi.readObject();
                        if( personnel != null ){
                            personnelVector.addElement(personnel);
                        }
                    }
                    oi.close();
                    fileInput.close();
                }
            }
            catch( IOException io ){
                System.out.println(io);
            }
            catch( ClassNotFoundException c ){
                debug("ClassNotFoundException: "+c);
            }
        }
    }
    return personnelVector;
}

/**
 * Return a vector of all atomics in the selected project
 */
public Vector getAllAtomics(){
    File aProject = new File(this.pathName); //current project
    projectAtomicVector = new Vector();
    if( aProject.isDirectory() ){
        String[] stepList = aProject.list(); //List all the steps of the current project
        for( int i=0; i<stepList.length; i++ ){
            File aFile = new File(aProject,stepList[i]); //searching for steps only
            if( aFile.isDirectory() ){
                File temp = new File(aProject, stepList[i]);
                String[] theFiles = temp.list();
                for( int j=0; j<theFiles.length; j++ ){
                    getAtomics(new File(temp, theFiles[j]), projectAtomicVector);
                }
            }
        }
    }
    return projectAtomicVector;
}

/**
 * Save all stepContent objects after updating

```

```

*/
    public void saveStepContentVector(Vector v, Vector majorMinorJobs, String stepName){
        projectStepContentVector = v;
        if( projectStepContentVector != null ){
            for( int i=0; i<projectStepContentVector.size(); i++ ){
                try{
                    String path = (String)stepContentPathVector.elementAt(i);
                    StepContent stepContent = (StepContent)projectStepContentVector.elementAt(i);
                    if( (stepContent.getStepName()).equals(stepName) ){
                        int lastSlash = (int) path.lastIndexOf("\\");
                        String stepPath = (String)path.substring(0,lastSlash);
                        File f = new File(stepPath+"\\\\"+COMPONENT_CONTENT_DIR+"\\\\"+DATA_LINK);
                        FileWriter fw = new FileWriter(f);
                        BufferedWriter bw = new BufferedWriter( fw);
                        if( bw != null ){
                            for( int j=0; j<majorMinorJobs.size(); j++ ){
                                Personnel personnel = (Personnel)majorMinorJobs.elementAt(j);
                                debug("personnel id : "+personnel.getID());
                                File theFile = new File(STAKEHOLDER+personnel.getID());
                                debug("personnel getAbsolutePath : "+theFile.getAbsolutePath());
                                bw.write(theFile.getAbsolutePath()+"\n");
                            }
                        }
                        bw.flush();
                        bw.close();
                        fw.close();
                    }
                    FileOutputStream fileOutput = new FileOutputStream(path);
                    ObjectOutputStream oo = new ObjectOutputStream(fileOutput);
                    if( oo != null ){
                        oo.writeObject(stepContent);
                    }
                    oo.flush();
                    oo.close();
                    fileOutput.close();
                }
                catch(IOException io){
                    debug("IOException: "+io);
                }
            }
        }
    }

/**
 * Return a vector of stepContent objects in the selected project
 */
    public Vector getStepContentVector(){
        int stepContentIndex = 0;
        stepContentPathVector = new Vector();
        projectAtomicVector = (Vector)getAllAtomics();
        projectStepContentVector = new Vector();

        if( projectAtomicVector != null ){
            for( int j=0; j<projectAtomicVector.size(); j++ ){
                File theFile = new File((File)projectAtomicVector.elementAt(j),"step.cnt"); //Example to get the atomic at element 4 of
projectAtomicVector
                if( theFile.exists() ){
                    try{
                        FileInputStream fileInput = new FileInputStream(theFile);
                        ObjectInputStream oo = new ObjectInputStream(fileInput);
                        if( oo != null ){
                            StepContent st = (StepContent)oo.readObject();
                            if( st!= null ){
                                projectStepContentVector.insertElementAt(st, stepContentIndex); //Step content vector of all atomics in the
project
                                stepContentPathVector.insertElementAt(theFile.getAbsolutePath(), stepContentIndex++);
                            }
                        }
                    }
                }
            }
        }
    }

```

```

        oo.close();
        fileInput.close();
    }
    catch(IOException io){
        debug("IOException: "+io);
    }
    catch( ClassNotFoundException c ){
        debug("ClassNotFoundException: "+c);
    }
}
}
}

return projectStepContentVector;
}

/**
 * Return a vector of atomics with the status is "Scheduled"
 */
public Vector getScheduledAtomicVector(){
    scheduledAtomicVector = new Vector();
    if( projectStepContentVector != null ){
        for( int i=0; i<projectStepContentVector.size(); i++ ){
            StepContent st = (StepContent)projectStepContentVector.elementAt(i);
            if( st != null ){
                String status = ((String)st.getStatus()).trim();
                if( status.equals("Approved") ){
                    st.setStatus("Scheduled");
                    scheduledAtomicVector.addElement(st);
                }
                else if( status.equals("Scheduled") ){
                    scheduledAtomicVector.addElement(st);
                }
            }
        }
    }
    return scheduledAtomicVector;
}

void checking_person(){
    for(int i=0; i<person_queue.size(); i++){
        Personnel person=(Personnel)person_queue.elementAt(i);
        if(person!=null){
            Vector major=(Vector) person.getMajorJobs();

            if(major!=null){
                for(int j=0; j<major.size(); j++){
                    if(major!=null){
                        StepContent step=(StepContent) major.elementAt(j);
                        if(step!=null){
                            if( check_status(step.getStepName())==1){
                                major.removeElement(step);
                            }
                        }
                    }
                }
            }
        }
    }
}

//check minorJob second
Vector minor=(Vector) person.getMinorJobs();
if(minor!=null){
    for(int k=0; k<minor.size(); k++){
        StepContent step=(StepContent) minor.elementAt(k);
        if(step!=null){
            if( check_status(step.getStepName())==1){
                minor.removeElement(step);
            }
        }
    }
}

```

```

        }//end for_loop(k)
    }
    }//end if(person!=null)

    }// end for_loop(i)
} //end check_person

int check_status(String stepname){
    for(int i=0; i<job_pool.size(); i++){
        if(job_pool!=null){
            StepContent step=(StepContent) job_pool.elementAt(i);
            if(step!=null){
                if(step.getStepName().equals(stepname)){
                    if(step.getStatus().equals("Completed")){
                        return 1;
                    }
                }
            }
        }
    }
} //end for_loop
return 0;
}

void cleanperson_job(){
    try{
        for(int i=0; i<person_queue.size(); i++){
            Personnel p=(Personnel) person_queue.elementAt(i);
            Vector v1=p.getMajorJobs();
            Vector v2=p.getMinorJobs();

            v1.removeAllElements();
            p.setMajorJobs(v1);
            v2.removeAllElements();
            p.setMinorJobs(v2);
            int personid=Integer.parseInt(p.getID());
            setperson_queue(personid, p);
        }
        savePersonnelVector(person_queue);
    }
    catch(Exception e){}
}

void setperson_queue(int personid, Personnel p1){
    try{
        for(int i=0; i<person_queue.size(); i++){
            Personnel p=(Personnel) person_queue.elementAt(i);
            if(personid==Integer.parseInt(p.getID())){
                person_queue.setElementAt(p1, i);
            }
        }
    } //end for_loop
}

catch(Exception e){}
}

void setstep(String stepname, StepContent step1){
    for(int i=0; i<job_pool.size(); i++){
        StepContent step=(StepContent) job_pool.elementAt(i);
        if(step.getStepName().equals(stepname)){
            job_pool.setElementAt(step1, i);
        }
    }
}

}

////////////////////////////////// END JOB SCHEDULE //////////////////////////////////
}

```


2. Cases.CasesTitle

```
/**-----
 * Filename: CasesTitle.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: JDK 1.3.1
 * Description: contains all global variables in Cases package. All the titles,
 * directories, or the other names are located in this class
 * Modified to include ReqPro, IE explorer, and graph panel variables and constraints
 *-----
 */
package Cases;

import java.io.File;

/**
 * CasesTitle : contains all global variables in Cases package.
 * All the titles, directories, or the other names are located in this class
 */
public interface CasesTitle {

    //Locations of Cases project files
    static final File CASESDIRECTORY = new File("\\CASES\\data\\cases\\");
    //Location of CSV files
    static final File CSVDIRECTORY = new File("\\CASES\\data\\CSV\\");
    //Location of stakeholder files
    static final File STAKEHOLDER = new File("\\CASES\\data\\stakeholder\\");

    //Locations of iexplorer, notepad, winword, excel, caps, and requisitepro
    static final String NOTEPAD = "notepad.exe";
    static final String WINWORD = "C:\\Program Files\\Microsoft Office\\Office\\Winword.exe";
    static final String EXCEL = "C:\\Program Files\\Microsoft Office\\Office\\Excel.exe";
    static final String IEXPLORER = "C:\\Program Files\\Internet Explorer\\iexplore.exe";
    static final String CAPS = "caps.bat";
    static final String REQPRO = "C:\\Program Files\\Rational\\RequisitePro\\bin\\requisitepro.exe";
    static final String[] EXECUTIONS = {NOTEPAD, WINWORD, EXCEL, null, IEXPLORER, CAPS, REQPRO};

    //Names of files which Cases will generate
    static final String COMPONENT_CFG = "component.cfg";
    static final String CURRENT_VSN = "current.vsn";
    static final String DEPENDENCY_CFG = "dependency.cfg";
    static final String LOOP_CFG = "loop.cfg";
    static final String STEP_CFG = "step.cfg";

    //Titles of frames under Spider menu
    static final String EDIT_TITLE = "SPIDER-Edit";
    static final String DECOMPOSE_TITLE = "SPIDER-Decompose";
    static final String COMPONENT_CONTENT_TITLE = "SPIDER-Component Content";
    static final String STEP_CONTENT_TITLE = "SPIDER-Step Content";
    static final String TRACE_TITLE = "SPIDER-Trace";

    //title of combo boxes
    static final String PROCESS_TITLE = "Select a Process";
    static final String STEP_TYPE_TITLE = "Select a Step Type";
    static final String COMPONENT_TYPE_TITLE = "Select a Component Type";
    static final String LINKS_TITLE = "Select a Link Type";
    static final String SKILL_TITLE = "ID : Name : Level";
    static final String[] BUTTONS = {"Add", "Edit", "Delete"};

    //Links file name inside Component Content directory (COMPONENT_CONTENT_DIR)
    static final String COMPONENT_CONTENT_DIR = "Component Content";
    static final String DATA_LINK = "data.link";
    static final String[] LINK_FILE_NAMES = {"txt.link", "word.link", "excel.link", DATA_LINK, "url.link",
"caps.link", "reqpro.link"};

    //two types of variant in EHL merging frame
```

```

static final String[] VARIANT_TYPES = {"Old", "New"};

//data is using for SkillTable
static final String[] STATUS = {"Proposed", "Approved", "Scheduled", "Assigned", "Decomposed", "Abandoned", "Completed"};
static final int SKILL_LEVEL = 4; //from 0 to 3
static final int SKILL_ID = 21; //from 1 to 20
static final int SECURITY_LEVEL = 6; //from 0 to 5
static final int PRIORITY_LEVEL = 6; //from 0 to 5
static final String[] SKILL_LIST = {"Unix System", "CAPS", "TAE Plus", "C", "C++",
    "Ada", "Notepad", "MS Word", "MS Excel", "Rational Rose",
    "UML", "System Analysis", "System Design", "Coding", "Testing",
    "Maintenance", "Organization", "Evaluation", "Management", "Negotiation"}; //20 skills name

// data is used for GraphPanel
static final int MAXCOMPONENTS = 10; // too many components can severely effect performance
static final int MAX = MAXCOMPONENTS + 1;
static final int COMPONENTSIZE = 50;
static final int NAMESIZE = 50;
static final int COMPONENTTRADIX = 35;
static final int DEFAULT_WEIGHT = 50;
static final int MAXDEPENDENCIES = 10; // too many dependencies can severely effect performance
}

```

3. Cases.ComponentContentFrame

```

/**-----
 * Filename: ComponentContentFrame.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: JDK 1.3.1
 * Description: ComponentContentFrame : create/edit/delete Component Content directory
 * with all link files for a selected step.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_ComponentContent
 *-----
 */
package Cases;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.*;
import java.util.StringTokenizer;
import java.util.Vector;
import javax.swing.*;
import Cases.Interfaces.I_ComponentContent;

/** ComponentContentFrame : create/edit/delete Component Content directory with all link
 * files for a selected step.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_ComponentContent
 */
public class ComponentContentFrame extends JFrame implements CasesTitle, I_ComponentContent, ActionListener, ItemListener
{
    public Vector fnList = new Vector();
    public String pathName = CASESDIRECTORY.getAbsolutePath();
    public String currentComponent = null;

    /**
     * Create ComponentContentFrame

```

```

*/
    public ComponentContentFrame() {
        initGUI();
    }

/**
 * method for initializing the GUI and its componenents
 */
private void initGUI() {
    //{{INIT_CONTROLS

    ReqProComboBox.setBounds(new java.awt.Rectangle(126, 273, 500, 20));
    jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    jLabel1.setText("ReqPro Files:");
    jLabel1.setForeground(java.awt.Color.black);
    jLabel1.setBounds(new java.awt.Rectangle(36, 273, 90, 20));
    jLabel9.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
    jLabel9.setText("Component Content Links");
    jLabel9.setForeground(java.awt.Color.black);
    jLabel9.setFont(new Font("Dialog", Font.BOLD, 15));
    jLabel9.setBounds(new java.awt.Rectangle(247, 13, 190, 24));
    jLabel7.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    jLabel7.setText("MS Word Files:");
    jLabel7.setForeground(java.awt.Color.black);
    jLabel7.setBounds(new java.awt.Rectangle(35, 82, 90, 20));
    jLabel8.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    jLabel8.setText("MS Excel Files:");
    jLabel8.setForeground(java.awt.Color.black);
    jLabel8.setBounds(new java.awt.Rectangle(35, 122, 90, 20));
    textComboBox.setBounds(new java.awt.Rectangle(125, 42, 500, 20));
    URLComboBox.setBounds(new java.awt.Rectangle(125, 202, 500, 20));
    MSExcelComboBox.setBounds(new java.awt.Rectangle(125, 122, 500, 20));
    jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    jLabel1.setText("CAPS Files:");
    jLabel1.setForeground(java.awt.Color.black);
    jLabel1.setBounds(new java.awt.Rectangle(35, 242, 90, 20));
    MSWordComboBox.setBounds(new java.awt.Rectangle(125, 82, 500, 20));
    CAPSComboBox.setBounds(new java.awt.Rectangle(125, 242, 500, 20));
    jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    jLabel2.setText("Data Files:");
    jLabel2.setForeground(java.awt.Color.black);
    jLabel2.setBounds(new java.awt.Rectangle(35, 162, 90, 20));
    jLabel3.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    jLabel3.setText("URLs");
    jLabel3.setForeground(java.awt.Color.black);
    jLabel3.setBounds(new java.awt.Rectangle(35, 202, 90, 20));
    jLabel4.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    jLabel4.setText("Text Files:");
    jLabel4.setForeground(java.awt.Color.black);
    jLabel4.setBounds(new java.awt.Rectangle(35, 42, 90, 20));
    setTitle("SPIDER-Component Content");
    getContentPane().setLayout(null);
    setSize(650,520);
    setVisible(false);
    getContentPane().add(saveButton);
    saveButton.setBounds(0,0,0,0);
    getContentPane().add(editButton);
    editButton.setBounds(0,0,0,0);
    getContentPane().add(deleteButton);
    deleteButton.setBounds(0,0,0,0);
    getContentPane().add(addButton);
    addButton.setBounds(0,0,0,0);
    cancelButton.setText("Cancel");
    cancelButton.setActionCommand("Cancel");
    getContentPane().add(cancelButton);
    cancelButton.setBounds(564,450,73,24);
    jLabel5.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    jLabel5.setText("Component Content:");

```

```

        getContentPane().add(JLabel5);
        JLabel5.setForeground(java.awt.Color.black);
        JLabel5.setFont(new Font("Dialog", Font.BOLD, 16));
        JLabel5.setBounds(0,7,170,30);
        componentLabel.setText("Component Content:");
        getContentPane().add(componentLabel);
        componentLabel.setForeground(java.awt.Color.black);
        componentLabel.setFont(new Font("Dialog", Font.BOLD, 15));
        componentLabel.setBounds(175,7,500,30);
        JLabel6.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel6.setText("Available Components:");
        getContentPane().add(JLabel6);
        JLabel6.setForeground(java.awt.Color.black);
        JLabel6.setBounds(10,60,130,20);
        getContentPane().add(componentsComboBox);
        componentsComboBox.setBounds(140,60,500,20);
        getContentPane().add(connectButton);
        connectButton.setBounds(0,0,0,0);
        JPanel1.setBorder(lineBorder1);
    }

    JPanel1.setLayout(null);
    getContentPane().add(JPanel1);
    JPanel1.setBounds(new java.awt.Rectangle(4, 92, 630, 350));
    JPanel1.add(dataComboBox);
    dataComboBox.setBounds(new java.awt.Rectangle(125, 160, 500, 20));
    JLabel9.setBorder(lineBorder1);
}

    getContentPane().add(JPanel1);
    JPanel1.setFont(new Font("Dialog", Font.BOLD, 12));
    JPanel1.add(JLabel4);
    JPanel1.add(JLabel3);
    JPanel1.add(JLabel2);
    JPanel1.add(CAPSCombobox);
    JPanel1.add(MSWordComboBox);
    JPanel1.add(JLabel1);
    JPanel1.add(MSEExcelComboBox);
    JPanel1.add(URLComboBox);
    JPanel1.add(textComboBox);
    JPanel1.add(JLabel8);
    JPanel1.add(JLabel7);
    JPanel1.add(JLabel9);
    JPanel1.add(jLabel1);
    JPanel1.add(ReqProComboBox);
    //$ lineBorder1.move(0,521);
    //$ }

    //{{INIT_MENU
    //}}

    //{{REGISTER_LISTENERS

    addButton.addActionListener(this);
    editButton.addActionListener(this);
    deleteButton.addActionListener(this);
    saveButton.addActionListener(this);
    cancelButton.addActionListener(this);

    componentsComboBox.addItemListener(this);
    //$ }
}

/**
 * Constructor for CasesFrame to launch ComponentContentFrame
 */
public ComponentContentFrame( String projectName, String stepName, Vector itemVector, Vector fnList ){
    this();
    //Add Save button
    saveButton.setText("Save");
}

```

```

        saveButton.setActionCommand("Save");
        getContentPane().add(saveButton);
        saveButton.setBounds(492,450,70,24);

        //Add Edit button
        editButton.setText("Edit");
        editButton.setActionCommand("Edit");
        getContentPane().add(editButton);
        editButton.setBounds(81,450,70,24);

        //Add Delete button
        deleteButton.setText("Delete");
        deleteButton.setActionCommand("Delete");
        getContentPane().add(deleteButton);
        deleteButton.setBounds(152,450,70,24);

        //Add Add button
        addButton.setText("Add");
        addButton.setActionCommand("Add");
        getContentPane().add(addButton);
        addButton.setBounds(10,450,70,24);

        this.fnList = fnList;
        this.componentLabel.setText(stepName);
        this.pathName += "\\ " + projectName;
        this.componentsComboBox.addItem(COMONENT_TYPE_TITLE);
        for( int i=0; i<itemVector.size(); i++ ){
            this.componentsComboBox.addItem( itemVector.elementAt(i) );
        }
    }

    /**
     * Constructor for TraceFrame to launch ComponentContentFrame
     */
    public ComponentContentFrame( String projectName, String stepName, String currentItem, Vector fnList ){
        this();
        this.fnList = fnList;
        this.componentLabel.setText(stepName);
        this.pathName += "\\ " + projectName;
        this.componentsComboBox.addItem(currentItem);
        this.componentsComboBox.setSelectedItem(currentItem);
        this.componentsComboBox.setEnabled(false);

        //Connect button is used in trace panel
        connectButton.setText("Connect");
        getContentPane().add(connectButton);
        connectButton.setBounds(517,380,81,24);
    }

    /**
     * method to set the title of the frame to sTitle
     * @param String
     */
    public ComponentContentFrame(String sTitle)    {
        this();
        setTitle(sTitle);
    }

    /**
     * a method to control the visible of the frame based upon b
     * @param boolean
     */
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

```

```

/**
 * a main procedure for unit testing.
 */
    static public void main(String[] args)    {
        (new ComponentContentFrame()).setVisible(true);
    }

/**
 * this method overrides the super addNotify()
 */
    public void addNotify()    {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;
    //{{{DECLARE_CONTROLS

JComboBox URLComboBox = new javax.swing.JComboBox();
JComboBox textComboBox = new javax.swing.JComboBox();
JButton saveButton = new javax.swing.JButton();
JButton editButton = new javax.swing.JButton();
JButton deleteButton = new javax.swing.JButton();
JButton addButton = new javax.swing.JButton();
JButton cancelButton = new javax.swing.JButton();
JLabel JLabel5 = new javax.swing.JLabel();
JLabel componentLabel = new javax.swing.JLabel();
JLabel JLabel6 = new javax.swing.JLabel();
JComboBox componentsComboBox = new javax.swing.JComboBox();
JButton connectButton = new javax.swing.JButton();
JLabel JLabel8 = new javax.swing.JLabel();
JLabel JLabel7 = new javax.swing.JLabel();
JPanel JPanel1 = new javax.swing.JPanel();
JComboBox dataComboBox = new javax.swing.JComboBox();
JLabel JLabel9 = new javax.swing.JLabel();
JLabel JLabel4 = new javax.swing.JLabel();
JLabel JLabel3 = new javax.swing.JLabel();
JLabel JLabel2 = new javax.swing.JLabel();
JComboBox CAPSComboBox = new javax.swing.JComboBox();
JComboBox ReqProComboBox = new javax.swing.JComboBox();
JComboBox MSWordComboBox = new javax.swing.JComboBox();
JLabel JLabel11 = new javax.swing.JLabel();
JLabel JLabel1 = new javax.swing.JLabel();
JComboBox MSExcelComboBox = new javax.swing.JComboBox();

/*      com.symantec.itools.java.swing.borders.LineBorder lineBorder1 = new
com.symantec.itools.java.swing.borders.LineBorder();
*/

    //}}}

    //{{{DECLARE_MENUS
    //}}}

```

```

/**
 * method to control action performed events
 */
public void actionPerformed(ActionEvent event)
{
    Object object = event.getSource();
    if (object == addButton) // add button pressed
        addButton_actionPerformed(event);
    else if (object == editButton) // edit button pressed
        editButton_actionPerformed(event);
    else if (object == deleteButton) // delete button pressed
        deleteButton_actionPerformed(event);
    else if (object == saveButton) // save button pressed
        saveButton_actionPerformed(event);
    else if (object == cancelButton) // cancel button pressed
        cancelButton_actionPerformed(event);
}

/**
 * Launch ConnectionLinksFrame
 * Adding more connection links to a componentContent
 */
public void addButton_actionPerformed(ActionEvent event) {
    String s = (String)this.componentsComboBox.getSelectedItemAt();
    (new ConnectionLinksFrame(this, s, BUTTONS[0])).setVisible( true );
}

/**
 * Launch ConnectionLinksFrame
 * Editing existing connection links in a componentContent
 */
public void editButton_actionPerformed(ActionEvent event) {
    String s = (String)this.componentsComboBox.getSelectedItemAt();
    (new ConnectionLinksFrame(this, s, BUTTONS[1])).setVisible( true );
}

/**
 * Launch ConnectionLinksFrame
 * Delete connection links in a componentContent
 */
public void deleteButton_actionPerformed(ActionEvent event) {
    String s = (String)this.componentsComboBox.getSelectedItemAt();
    (new ConnectionLinksFrame(this, s, BUTTONS[2])).setVisible( true );
}

/**
 * Save all link files under Component Content directory of
 * currentComponent
 */
public void saveButton_actionPerformed(ActionEvent event) {
    for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
        saveLinkFile( LINK_FILE_NAMES[i] );
    }
    clearComboBoxes();
    setVisible(false);
    dispose();
}

/**
 * Clear all items in every combo box
 * Exit ComponentContentFrame
 */
public void cancelButton_actionPerformed(ActionEvent event) {
    clearComboBoxes();
    setVisible(false);
    dispose();
}

```

```

        public void itemStateChanged(ItemEvent event)    {
            Object object = event.getSource();
            if (object == componentsComboBox)
                componentsComboBox_itemStateChanged(event);
        }

/**
 * Select an item in the combo box
 */
public void componentsComboBox_itemStateChanged(ItemEvent event)  {
    String selectedItem = null;
    if( event.getStateChange() == event.SELECTED ){
        selectedItem = (String)event.getItem();
        if( !selectedItem.equals(ComponentType.TITLE) ){
            clearComboBoxes();
            setButtonEnabled( true );
            this.currentComponent = (String)checkInput(selectedItem);
            if( !this.currentComponent.equals("") ){
                File f = (File)searchPath(this.currentComponent);
                if( !f.exists() ){
                    JOptionPane.showMessageDialog(this, selectedItem+" does not exist!", "Alert Message",
                        JOptionPane.ERROR_MESSAGE);
                }
                else if( f.exists() ){
                    String[] list = f.list();
                    for( int j=0; j<list.length; j++ ){
                        String s = (String)list[j];
                        File aFile = null;
                        if( s.equals(ComponentType.CONTENT_DIR) ){
                            aFile = new File(f, s);
                            if( !aFile.isDirectory() ){
                                aFile.mkdir();
                                j = list.length;
                                setButtonEnabled( false );
                            }
                        }
                        else if( aFile.isDirectory() ){
                            j = list.length;
                            setButtonEnabled( true );
                            list = aFile.list();
                            for( int i=0; i<list.length; i++ ){
                                searchFiles(aFile, (String)list[i]);
                            }
                        }
                    }
                }
                else if( !s.equals(ComponentType.CONTENT_DIR) ){
                    aFile = new File(f, ComponentType.CONTENT_DIR);
                    aFile.mkdir();
                    setButtonEnabled( true );
                }
            }
        }
    }
    else if( selectedItem.equals(ComponentType.TITLE) ){
        setButtonEnabled( false );
    }
}

/**
 * Add/Delete/Edit/Save button only active when a step is selected
 */
public void setButtonEnabled( boolean flag ){
    addButton.setEnabled( flag );
    editButton.setEnabled( flag );
    deleteButton.setEnabled( flag );
}

```



```

        saveButton.setEnabled( flag );
    }

/**
 * Short cut to print the output
 * @param string : the output string
 */
void debug( String s ){
    System.out.println(s);
}

/**
 * Search and get the info from link files
 *
 * @param aFile : a link file
 * @param fileType : a name of link file
 */
public void searchFiles(File aFile, String fileType){
File f = new File( aFile, fileType);
try{
    BufferedReader br = null;
    String item = null;
    if( fileType.equals(LINK_FILE_NAMES[0]) ){
        br = new BufferedReader( new FileReader(f));
        if( br != null ){
            while( (item = br.readLine()) != null ){
                this.textComboBox.addItem(item);
            }
        }
    }
    else if( fileType.equals(LINK_FILE_NAMES[1]) ){
        br = new BufferedReader( new FileReader(f));
        if( br != null ){
            while( (item = br.readLine()) != null ){
                this.MSWordComboBox.addItem(item);
            }
        }
    }
    else if( fileType.equals(LINK_FILE_NAMES[2]) ){
        br = new BufferedReader( new FileReader(f));
        if( br != null ){
            while( (item = br.readLine()) != null ){
                this.MSExcelComboBox.addItem(item);
            }
        }
    }
    else if( fileType.equals(LINK_FILE_NAMES[3]) ){
        br = new BufferedReader( new FileReader(f));
        if( br != null ){
            while( (item = br.readLine()) != null ){
                this.dataComboBox.addItem(item);
            }
        }
    }
    else if( fileType.equals(LINK_FILE_NAMES[4]) ){
        br = new BufferedReader( new FileReader(f));
        if( br != null ){
            while( (item = br.readLine()) != null ){
                this.URLComboBox.addItem(item);
            }
        }
    }
    else if( fileType.equals(LINK_FILE_NAMES[5]) ){
        br = new BufferedReader( new FileReader(f));
        if( br != null ){
            while( (item = br.readLine()) != null ){
                this.CAPSCComboBox.addItem(item);
            }
        }
    }
}

```

```

    }
    }
    else if( fileType.equals(LINK_FILE_NAMES[6]) ){
        br = new BufferedReader( new FileReader(f));
        if( br != null ){
            while( (item = br.readLine()) != null ){
                this.ReqProComboBox.addItem(item);
            }
        }
    }
}
}
catch( IOException io ){
    debug("IOException: "+io);
}
}

/**
 * Form an absolute path for a selected component
 *
 * @param s : a selected component name
 * @return File which is formed by a string s
 */
public File searchPath( String s){
    String thePath = null;
    char tempChar = s.charAt(0);
    if (tempChar != 'a')
        s = "s-"+((char)(tempChar-1))+s;
    else
        s = "s-"+s;

    //To convert the string of selected item into the file path
    for( int i=0; i<this.fnList.size(); i++ ){
        String fn = (String)this.fnList.elementAt(i);
        String sub = s.substring(0,fn.length());
        if( sub.equals(fn)){
            i = this.fnList.size();
            thePath= sub+"\\\\";
            String sub2 = s.substring(fn.length());
            char[] ca = (char[])sub2.toCharArray();
            int result = 0;
            for( int j=0; j<ca.length; j++ ){
                String s3 = ca[j]+"";
                result = s3.compareTo("-");
                if( result == 0 ){
                    j=ca.length;
                    StringTokenizer st = new StringTokenizer(sub2,"-");
                    String before_ = st.nextToken("-");
                    thePath = thePath+before_;
                    String _after = st.nextToken("-");
                    st = new StringTokenizer(_after,".");
                    while(st.hasMoreTokens()){
                        thePath = thePath+"\\\\"+st.nextToken();
                    }
                }
                else if( (result > 0) && (j==ca.length - 1) ){
                    thePath = thePath+sub2;
                }
            }
        }
    }
    debug("ComponentContentFrame:pathName:");debug(this.pathName);
    debug("ComponentContentFrame:thePath:");debug(thePath);
    File f = new File(this.pathName,thePath);
    return f;
}

/**

```

```

    * Set text file names in textComboBox
    *
    * @param v : a vector of links
    */
public void setTextLink(Vector v){
    this.textComboBox.removeAllItems();
    for( int i=0; i<v.size(); i++){
        this.textComboBox.addItem(v.elementAt(i));
    }
}

/**
 * Set word file names in MSWordComboBox
 *
 * @param v : a vector of links
 */
public void setWordLink(Vector v){
    this.MSWordComboBox.removeAllItems();
    for( int i=0; i<v.size(); i++){
        this.MSWordComboBox.addItem(v.elementAt(i));
    }
}

/**
 * Set excel file names in MSEXcelComboBox
 *
 * @param v : a vector of links
 */
public void setExcelLink(Vector v){
    this.MSEXcelComboBox.removeAllItems();
    for( int i=0; i<v.size(); i++){
        this.MSEXcelComboBox.addItem(v.elementAt(i));
    }
}

/**
 * Set personnel object names in dataComboBox
 *
 * @param v : a vector of links
 */
public void setDataLink(Vector v){
    this.dataComboBox.removeAllItems();
    for( int i=0; i<v.size(); i++){
        this.dataComboBox.addItem(v.elementAt(i));
    }
}

/**
 * Set URL in URLComboBox
 *
 * @param v : a vector of links
 */
public void setURLLink(Vector v){
    this.URLComboBox.removeAllItems();
    for( int i=0; i<v.size(); i++){
        this.URLComboBox.addItem(v.elementAt(i));
    }
}

/**
 * Set caps file names in CAPSComboBox
 *
 * @param v : a vector of links
 */
public void setCAPSLink(Vector v){
    this.CAPSComboBox.removeAllItems();
    for( int i=0; i<v.size(); i++){
        this.CAPSComboBox.addItem(v.elementAt(i));
    }
}

```

```

    }
}
/**
 * Set ReqPro file names in ReqProComboBox
 *
 * @param v : a vector of links
 */
public void setReqProLink(Vector v){
    this.ReqProComboBox.removeAllItems();
    for( int i=0; i<v.size(); i++ ){
        this.ReqProComboBox.addItem(v.elementAt(i));
    }
}
/**
 * Save all link files
 *
 * @param fileType : a type of file, e.g., txt.link, data.link, ...
 */
public void saveLinkFile( String fileType){
    Vector items = new Vector();
    JComboBox comboBox = (JComboBox)findComboBox( fileType );
    for( int i=0; i<comboBox.getItemCount(); i++ ){
        items.addElement(comboBox.getItemAt(i));
    }

    File f = (File)searchPath(this.currentComponent);
    f = new File(f.getAbsolutePath()+"\\"+COMPONENT_CONTENT_DIR+"\\"+fileType);
    try{
        FileWriter fw = new FileWriter(f);
        BufferedWriter bw = new BufferedWriter( fw);
        if( bw != null ){
            for( int i=0; i<comboBox.getItemCount(); i++ ){
                bw.write((String)comboBox.getItemAt(i)+"\n");
            }
        }
        bw.flush();
        bw.close();
        fw.close();
    }
    catch( IOException io ){
        debug("IOException: "+io);
    }
}

/**
 * Search a combobox matches with fileType
 *
 * @param fileType : a type of file, e.g., txt.link, data.link, ...
 * @return JComboBox : a combo box of fileType
 */
public JComboBox findComboBox( String fileType ){
    JComboBox comboBox = null;
    if( fileType.equals(LINK_FILE_NAMES[0]) ){
        comboBox = this.textComboBox;
    }
    else if( fileType.equals(LINK_FILE_NAMES[1]) ){
        comboBox = this.MSWordComboBox;
    }
    else if( fileType.equals(LINK_FILE_NAMES[2]) ){
        comboBox = this.MSExcelComboBox;
    }
    else if( fileType.equals(LINK_FILE_NAMES[3]) ){
        comboBox = this.dataComboBox;
    }
    else if( fileType.equals(LINK_FILE_NAMES[4]) ){
        comboBox = this.URLComboBox;
    }
    else if( fileType.equals(LINK_FILE_NAMES[5]) ){

```

```

        comboBox = this.CAPSComboBox;
    }
    else if( fileType.equals(LINK_FILE_NAMES[6]) ){
        comboBox = this.ReqProComboBox;
    }
    return comboBox;
}

/**
 * Refresh all comboboxes
 */
public void clearComboBoxes(){
    textComboBox.removeAllItems();
    MSWordComboBox.removeAllItems();
    MSExcelComboBox.removeAllItems();
    dataComboBox.removeAllItems();
    URLComboBox.removeAllItems();
    CAPSComboBox.removeAllItems();
    ReqProComboBox.removeAllItems();
}

/**
 * Check the selected component is valid or not
 *
 * @return String : null/valid component name
 * @param selectedItem : a selected component in componentComboBox
 */
public String checkInput(String selectedItem ){
    if( selectedItem != null ){
        int j = selectedItem.indexOf("-");
        if( j>0 ){
            String sub2 = (selectedItem.substring(j+1)).trim();
            if( !sub2.equals(this.componentLabel.getText()) ){
                char c = (char)sub2.charAt(0);
                boolean isLetter = (new Character(c)).isLetter(c);
                if( isLetter ){
                    JOptionPane.showMessageDialog(this, selectedItem+" is invalid component!",
                        "Error Message",JOptionPane.ERROR_MESSAGE);

                    return null;
                }
            }
            else{
                return this.componentLabel.getText();
            }
        }
    }
    return selectedItem;
}
}

```

4. Cases.ComponentType

```

/**-----
 * Filename: ComponentType.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: JDK 1.3.1
 * Description: Create a Component Type object and save it in component.cfg file
 * extended with QFD data and observable
 *-----
 */
package Cases;

```

```

import java.awt.*;
import java.io.Serializable;
import java.util.Hashtable;
import java.util.Observable;

/**
 * ComponentType : Create a Component Type object and save it in component.cfg file
 */

public class ComponentType extends Observable implements Serializable {

    /**
     * componentID : component type ID
     */
    private String componentID;

    /**
     * componentName : component type name
     */
    private String componentName;

    /**
     * componentDescription : component type description
     */
    private String componentDescription;

    /**
     * componentValue : integer value in array for node drawing
     */
    private int componentValue;

    /**
     * componentPosition : stores the location of node in GraphPanel
     */
    private Point componentPosition;
    /** a hashtable to store component data */
    private Hashtable componentHashtable=new Hashtable();

    /**
     * This ComponentType constructor is used to create a ComponentType object
     */
    public ComponentType( String componentID, String componentName, String componentDescription, int componentValue, Point
componentPosition ){
        this.componentID = componentID;
        this.componentName = componentName;
        this.componentDescription = componentDescription;
        this.componentValue = componentValue;
        this.componentPosition = componentPosition;
    }
    /**
     * default constructor
     */
    public ComponentType() {}

    /**
     * returns the component id
     * @return String
     */
    public String getComponentID(){
        return this.componentID;
    }
    /**
     * sets the component id to newID and
     * notifies the observers of changes
     * @param String newID
     */
    public void setComponentID(String newID) {
        this.componentID = newID;
        // must call setChanged before notifyObservers to

```

```

        // indicate model has changed
        setChanged();

        // notify Observers that model has changed
        notifyObservers();
    }

    /**
     * returns the component name
     * @return componentName
     */
    public String getComponentName(){
        return this.componentName;
    }

    /**
     * method to set component name to new name and
     * notifies observers of changes
     * @param String newName
     */
    public void setComponentName(String newName) {
        this.componentName = newName;
        // must call setChanged before notifyObservers to
        // indicate model has changed
        setChanged();

        // notify Observers that model has changed
        notifyObservers();
    }

    /**
     * returns the component description
     * @return componentDescription
     */
    public String getComponentDescription(){
        return this.componentDescription;
    }

    /**
     * sets the component description to newDescription
     * and notifies observers of changes
     * @param String newDescription
     */
    public void setComponentDescription (String newDescription) {
        this.componentDescription = newDescription;
        // must call setChanged before notifyObservers to
        // indicate model has changed
        setChanged();

        // notify Observers that model has changed
        notifyObservers();
    }

    /**
     * returns the component value
     * @return int componentValue
     */
    public int getComponentValue() {
        return this.componentValue;
    }

    /**
     * sets the component value to newValue
     * and notifies observers of changes
     * @param int newValue
     */
    public void setComponentValue(int newValue) {
        this.componentValue = newValue;
        // must call setChanged before notifyObservers to
        // indicate model has changed
        setChanged();

        // notify Observers that model has changed
        notifyObservers();
    }

```

```

    }
    /**
     * returns the component position
     * @return Point component position
     */
    public Point getComponentPosition() {
        return this.componentPosition;
    }
    /**
     * sets the component position to newPosition
     * and notifies observers
     * @param Point newPosition
     */
    public void setComponentPosition(Point newPosition) {
        this.componentPosition = newPosition;
        // must call setChanged before notifyObservers to
        // indicate model has changed
        setChanged();

        // notify Observers that model has changed
        notifyObservers();
    }

    /**
     * returns componentHashtable
     * @return Hashtable
     */
    public Hashtable getComponentHashtable(){ return componentHashtable; }

    /**
     * set the component hashtable to property
     * @param Hashtable
     */
    public void setComponentHashtable(Hashtable property){ this.componentHashtable = property; }
}

```

5. Cases.ConnectionLinksFrame

```

/**-----
 * Filename: ConnectionLinksFrame.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: JDK 1.3.1
 * Description: Create/Edit/Delete links of a selected component
 *   Launched by ComponentContentFrame's buttons(Add, Edit, or Delete)
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Added standard listeners and handlers.
 *-----
 */
package Cases;

import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.ItemListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.Vector;
import javax.swing.*;

////////////////////////////////////
/**
 * ConnectionLinksFrame : Create/Edit/Delete links of a selected component
 * Launched by ComponentContentFrame's buttons(Add, Edit, or Delete)
 *

```



```

* Implement CasesTitle where stores all global variables of Cases package
*/
////////////////////////////////////
public class ConnectionLinksFrame extends javax.swing.JFrame implements CasesTitle, ActionListener, ItemListener, MouseListener
{
    /**
     * fd: instantiate an open fileDialog
     */
    FileDialog fd = new FileDialog(this, "Open",FileDialog.LOAD);

    /**
     * update : to check that all the changes is update yet
     */
    boolean update = true;

    /**
     * compContentFrame : get the current ComponentContentFrame
     */
    ComponentContentFrame compContentFrame = null;

    /**
     * listModel : to monitor item list
     */
    DefaultListModel listModel = new DefaultListModel();

    /**
     * selectedItem : current link type
     */
    private String selectedItem = null;

    /**
     * currentPosition : position of selected item in the itemList
     */
    private int currentPosition = 0;

    /**
     * button : name of button of ComponentContentFrame to launch ConnectionLinksFrame
     */
    String button = null;

    /**
     * storedVector : an array of vector of all links
     */
    Vector[] storedVector = {new Vector(), new Vector(), new Vector(), new Vector(), new Vector(), new Vector(), new Vector()};

    /**
     * Build ConnectionLinksFrame
     */
    /*      public ConnectionLinksFrame()  {
            initGUI();
        }
    */
    /**
     * method to initialize the GUI and its components
     */
    private void initGUI()      {

        //{{ INIT_CONTROLS
        setTitle("Component Content Editor");
        getContentPane().setLayout(null);
        setSize(500,450);
        setVisible(false);
        titleLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        titleLabel.setText("jlabel");
        getContentPane().add(titleLabel);
        titleLabel.setForeground(java.awt.Color.black);
        titleLabel.setFont(new Font("Dialog", Font.BOLD, 18));

```

```

titleLabel.setBounds(25,5,450,30);
exitButton.setText("Exit");
exitButton.setActionCommand("Cancel");
getContentPane().add(exitButton);
exitButton.setBounds(260,400,75,22);
itemScrollPane.setOpaque(true);
getContentPane().add(itemScrollPane);
itemScrollPane.setBounds(15,165,470,200);
itemScrollPane.getViewport().add(itemList);
itemList.setBounds(0,0,467,197);
updateButton.setText("Update");
updateButton.setActionCommand("OK");
getContentPane().add(updateButton);
updateButton.setBounds(164,400,75,22);
getContentPane().add(connectionComboBox);
connectionComboBox.setBounds(115,60,270,22);
JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
JLabel1.setText("Connection Links");
getContentPane().add(JLabel1);
JLabel1.setForeground(java.awt.Color.black);
JLabel1.setBounds(15,140,470,22);
getContentPane().add(tempTextField);
tempTextField.setBounds(12,100,321,22);
tempButton.setText("jbutton");
tempButton.setActionCommand("jbutton");
getContentPane().add(tempButton);
tempButton.setBounds(333,100,75,22);
browseButton.setText("Browse");
browseButton.setActionCommand("Browse");
getContentPane().add(browseButton);
browseButton.setBounds(408,100,78,22);
//}}

//{{INIT_MENUS
//}}

//{{REGISTER_LISTENERS

connectionComboBox.addItemListener(this);

tempButton.addActionListener(this);
updateButton.addActionListener(this);
exitButton.addActionListener(this);

itemList.addMouseListener(this);
browseButton.addActionListener(this);
//}}

/**
 * set default model for itemList
 */
itemList.setModel(listModel);

/**
 * create a combobox with all type of links
 */
for( int i=0; i<LINK_FILE_NAMES.length; i++){
    connectionComboBox.addItem(LINK_FILE_NAMES[i]);
}
}

/**
 * Constructor is launched by buttons from ComponentContentFrame
 */
public ConnectionLinksFrame(ComponentContentFrame compContentFrame, String title, String button){
//    this();

    initGUI();

```

```

        this.compContentFrame = compContentFrame;
        this.button = button;
        this.titleLabel.setText(title);
        this.tempButton.setText(button);
        setListModel();
    }
    /**
     * constructor to initial frame with sTitle
     * @param String
     */
    public ConnectionLinksFrame(String sTitle) {
        initGUI();
        setTitle(sTitle);
    }

    /**
     * a method to control the frames visibility based on b
     * @param boolean
     */
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

    /**
     * overrides the super method addNotify()
     */
    public void addNotify() {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    //{{DECLARE_CONTROLS
    javax.swing.JLabel titleLabel = new javax.swing.JLabel();
    javax.swing.JButton exitButton = new javax.swing.JButton();
    javax.swing.JScrollPane itemScrollPane = new javax.swing.JScrollPane();
    javax.swing.JList itemList = new javax.swing.JList();
    javax.swing.JButton updateButton = new javax.swing.JButton();
    javax.swing.JComboBox connectionComboBox = new javax.swing.JComboBox();
    javax.swing.JLabel JLabel1 = new javax.swing.JLabel();
    javax.swing.JTextField tempTextField = new javax.swing.JTextField();
    javax.swing.JButton tempButton = new javax.swing.JButton();
    javax.swing.JButton browseButton = new javax.swing.JButton();
    //}}

    //{{DECLARE_MENUS
    //}}

```

```

/**
 * an method to control item state changes for connection
 * comboboxes
 */
    public void itemStateChanged(java.awt.event.ItemEvent event)    {
        Object object = event.getSource();
        if (object == connectionComboBox)
            connectionComboBox_itemStateChanged(event);
    }

/**
 * Monitor what link type is selected
 */
    void connectionComboBox_itemStateChanged(java.awt.event.ItemEvent event)    {
        if( event.getStateChange() == event.SELECTED ){
            listModel.removeAllElements();
            this.tempTextField.setText("");
            selectedItem = (String)event.getItem();
            for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
                if( selectedItem.equals(LINK_FILE_NAMES[i]) ){
                    for( int j=0; j<storedVector[i].size(); j++ ){
                        listModel.addElement(storedVector[i].elementAt(j));
                    }
                    i= LINK_FILE_NAMES.length;
                }
            }
        }
    }

/**
 * a new method to control standard action events
 */
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == tempButton) // temp button pressed
            tempButton_actionPerformed(event);
        else if (object == updateButton) // update button pressed
            updateButton_actionPerformed(event);
        else if (object == exitButton) // exit button pressed
            exitButton_actionPerformed(event);
        else if (object == browseButton) // browse button pressed
            browseButton_actionPerformed(event);
    }

/**
 * method to handle temp button pressed event
 */
    void tempButton_actionPerformed(java.awt.event.ActionEvent event)    {
        String s = tempTextField.getText();
        if( !s.equals("") ){
            searchButton(s);
            this.tempTextField.setText("");
            update = false;
        }
        else{
            update = true;
        }
    }

/**
 * Update all link comboboxes in ComponentContentFrame
 */
    void updateButton_actionPerformed(java.awt.event.ActionEvent event)    {
        int index = (int)connectionComboBox.getSelectedIndex();
        searchLink(index);
        String s = (String)tempTextField.getText();
    }

```

```

        if( !s.equals("") ){
            searchButton(s);
        }
        update = true;
    }

/**
 * Confirmation dialog to give a user one more chance to update all
 * the change, and exit ConnectionLinksFrame.
 */
    void exitButton_actionPerformed(java.awt.event.ActionEvent event)    {
        if( !update ){
            Object[] options = { "OK", "CANCEL" };
            int i = JOptionPane.showOptionDialog(this, "You did not update the last change yet. Would you like to update?",
            "Warning", JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE, null, options, options[0]);
            if( i==0 ){
                updateButton_actionPerformed(event);
            }
        }
        setVisible( false );
        dispose();
    }

/**
 * Connect to FileDialog to let user to select a file insteads of typing, and return the
 * entire path of the file to the textfield
 */
    void browseButton_actionPerformed(java.awt.event.ActionEvent event)    {
        fd.show();
        String fileName = fd.getFile();
        if( fileName != null ){
            tempTextField.setText(fd.getDirectory()+fileName);
        }
    }

    // standard mouse events that are currently unused
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
    public void mousePressed(MouseEvent e) {
    }
    public void mouseReleased(MouseEvent e) {
    }

/**
 * mouse clicked event method
 */
    public void mouseClicked(MouseEvent event)    {
        Object object = event.getSource();
        if (object == itemList)
            itemList_mouseClicked(event);
    }

/**
 * Monitor which item is selected
 */
    void itemList_mouseClicked(java.awt.event.MouseEvent event)
    {
        if( event.getClickCount() > 0 ) {
            String s = (String)this.itemList.getSelectedValue();
            this.tempTextField.setText(s);
            this.currentPosition = (int)this.itemList.getSelectedIndex();
        }
    }

/**
 * Search which button (Add/Delete/Edit) from ComponentContentFrame to
 * launch this frame.

```

```

*
* @param s : button title which is passed from ComponentContentFrame
*/
void searchButton(String s){
    for( int i=0; i<BUTTONS.length; i++){
        if( this.button.equals(BUTTONS[i]) ){
            searchVector(s,i);
            i=BUTTONS.length;
        }
    }
}

/**
* Search which comboBox in ComponentContentFrame needs to update
*
* @param index : an index of selected item in connectionComboBox
*/
void searchLink(int index){
    if( index == 0 ){
        this.compContentFrame.setTextLink(storedVector[0]);
    }
    else if( index == 1 ){
        this.compContentFrame.setWordLink(storedVector[1]);
    }
    else if( index == 2 ){
        this.compContentFrame.setExcelLink(storedVector[2]);
    }
    else if( index == 3 ){
        this.compContentFrame.setDataLink(storedVector[3]);
    }
    else if( index == 4 ){
        this.compContentFrame.setURLLink(storedVector[4]);
    }
    else if( index == 5 ){
        this.compContentFrame.setCAPSLink(storedVector[5]);
    }
    else if( index == 6 ){
        this.compContentFrame.setReqProLink(storedVector[6]);
    }
}

/**
* Set the content of a selected link file into the list
*/
void setListModel(){
    if( this.compContentFrame != null ){
        for( int j=0; j<LINK_FILE_NAMES.length; j++){
            JComboBox cb = (JComboBox)this.compContentFrame.findComboBox(LINK_FILE_NAMES[j]);
            int c = cb.getItemCount();
            for( int i=0; i<c; i++){
                storedVector[j].addElement(cb.getItemAt(i));
            }
        }
        for( int j=0; j<storedVector[0].size(); j++){
            listModel.addElement(storedVector[0].elementAt(j));
        }
    }
}

/**
* Search which vector is respective with the selected link type
*
* @param s : a link type
* @param index : the index of selected link type and it use to match with
* the element in array vector, storedVector.
*/
void searchVector(String s, int index){
    for( int i=0; i<LINK_FILE_NAMES.length; i++){

```

```

        if( selectedItem.equals(LINK_FILE_NAMES[i]) ){
            if( index==0 ){
                if( !listModel.contains(s) ){
                    listModel.addElement(s);
                    storedVector[i].addElement(s);
                }
                else{
                    JOptionPane.showMessageDialog(this, s+" is already in the list!",
                                                    "Warning Message", JOptionPane.ERROR_MESSAGE);
                }
            }
            else if( index==1 ){
                listModel.removeElementAt(this.currentPosition);
                listModel.insertElementAt(s,this.currentPosition);
                storedVector[i].removeElementAt(this.currentPosition);
                storedVector[i].insertElementAt(s,this.currentPosition);
            }
            else if( index==2 ){
                if( listModel.contains(s) ){
                    listModel.removeElementAt(this.currentPosition);
                    storedVector[i].removeElementAt(this.currentPosition);
                }
                else{
                    JOptionPane.showMessageDialog(this, s+" is not in the list!",
                                                    "Warning Message", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
        i= LINK_FILE_NAMES.length;
    }
}
}
}

```

6. Cases.DecomposeListDialog

```

/**-----
 * Filename: DecomposeListDialog.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: JDK 1.3.1
 * Description: a dialog stores all decomposed components of a selected step.
 * It is launched by decomposeComboBox of TraceFrame
 * It has 2 lists, components and decomposed components
 * Modified with standard events and listeners
 *-----
 */
package Cases;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.File;
import java.util.Vector;
import javax.swing.*;

////////////////////////////////////
/**
 * DecomposeListDialog : a dialog stores all decomposed components of a selected step.
 * It is launched by decomposeComboBox of TraceFrame
 * It has 2 lists, components and decomposed components
 */
////////////////////////////////////
public class DecomposeListDialog extends javax.swing.JDialog implements ActionListener,

```

```

MouseListener {
/**
 * Parent frame of this dialog
 */
TraceFrame tf = null;

/**
 * Monitor all components of the selected step
 */
DefaultListModel componentListModel = new DefaultListModel();

/**
 * Monitor all decomposed components of the selected component
 */
DefaultListModel decomposeListModel = new DefaultListModel();

/**
 * Build DecomposeListDialog
 */
    public DecomposeListDialog(JFrame parent)    {
        super(parent);

        //{{INIT_CONTROLS
        setModal(true);
        getContentPane().setLayout(null);
        setSize(585,400);
        setVisible(false);
        componentScrollPane.setOpaque(true);
        getContentPane().add(componentScrollPane);
        componentScrollPane.setBounds(15,90,270,230);
        componentScrollPane.getViewport().add(componentList);
        componentList.setBounds(0,0,267,227);
        OKButton.setText("OK");
        OKButton.setActionCommand("OK");
        getContentPane().add(OKButton);
        OKButton.setBounds(207,340,75,22);
        cancelButton.setText("Cancel");
        cancelButton.setActionCommand("Cancel");
        getContentPane().add(cancelButton);
        cancelButton.setBounds(303,340,75,22);
        titleLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        titleLabel.setText("JLabel");
        getContentPane().add(titleLabel);
        titleLabel.setForeground(java.awt.Color.black);
        titleLabel.setFont(new Font("Dialog", Font.BOLD, 18));
        titleLabel.setBounds(42,5,500,30);
        decomposeScrollPane.setOpaque(true);
        getContentPane().add(decomposeScrollPane);
        decomposeScrollPane.setBounds(300,90,270,230);
        decomposeScrollPane.getViewport().add(decomposeList);
        decomposeList.setBounds(0,0,267,227);
        JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        JLabel1.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
        JLabel1.setText("Component");
        getContentPane().add(JLabel1);
        JLabel1.setForeground(java.awt.Color.black);
        JLabel1.setBounds(15,68,270,22);
        JLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        JLabel2.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
        JLabel2.setText("Decompose");
        getContentPane().add(JLabel2);
        JLabel2.setForeground(java.awt.Color.black);
        JLabel2.setBounds(300,68,270,22);
        //}}

        //{{REGISTER_LISTENERS

        OKButton.addActionListener(this);

```



```

        cancelButton.addActionListener(this);

        componentList.addMouseListener(this);
    //}}

    /**
     * setModel for componentList and decomposeList
     */
    componentList.setModel(componentListModel);
    decomposeList.setModel(decomposeListModel);
}

/**
 * TraceFrame use this constructor to launch DecomposeListDialog
 */
public DecomposeListDialog(TraceFrame tf, String title, Vector itemVector)
{
    this((JFrame)tf);
    this.tf = tf;
    setTitle(title);
    this.titleLabel.setText(title);
    for( int i=0; i<itemVector.size(); i++){
        componentListModel.addElement(itemVector.elementAt(i));
    }
}

/**
 * a method to control the visibility of dialog based upon b
 * @param boolean
 */
public void setVisible(boolean b) {
    if (b)
        setLocation(50, 50);
    super.setVisible(b);
}

/**
 * overrides super addNotify() method
 */
public void addNotify() {
    // Record the size of the window prior to calling parents addNotify.
    Dimension size = getSize();

    super.addNotify();

    if (frameSizeAdjusted)
        return;
    frameSizeAdjusted = true;

    // Adjust size of frame according to the insets
    Insets insets = getInsets();
    setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height);
}

// Used by addNotify
boolean frameSizeAdjusted = false;

//{{DECLARE_CONTROLS
javax.swing.JScrollPane componentScrollPane = new javax.swing.JScrollPane();
javax.swing.JList componentList = new javax.swing.JList();
javax.swing.JButton OKButton = new javax.swing.JButton();
javax.swing.JButton cancelButton = new javax.swing.JButton();
javax.swing.JLabel titleLabel = new javax.swing.JLabel();
javax.swing.JScrollPane decomposeScrollPane = new javax.swing.JScrollPane();
javax.swing.JList decomposeList = new javax.swing.JList();
javax.swing.JLabel JLabel1 = new javax.swing.JLabel();
javax.swing.JLabel JLabel2 = new javax.swing.JLabel();

```

```

        //}}
    /**
     * tracks standard action events
     */
    public void actionPerformed(java.awt.event.ActionEvent event) {
        Object object = event.getSource();
        if (object == OKButton) // okay button pressed
            OKButton_actionPerformed(event);
        else if (object == cancelButton) // cancel button pressed
            cancelButton_actionPerformed(event);
    }

    /**
     * Return a selected decompose step to TraceFrame
     */
    void OKButton_actionPerformed(ActionEvent event) {
        String selectedItem = (String)this.decomposeList.getSelectedValue();
        if (selectedItem != null ){
            if (this.tf != null ){
                String s = this.tf.convertToThePath(selectedItem);
                if ( s!= null ){
                    this.tf.setSelectedItem(s, selectedItem);
                    setVisible(false);
                    dispose();
                }
            }
        }
        else{
            setVisible(false);
            dispose();
        }
    }

    /**
     * Exit DecomposeListDialog
     */
    void cancelButton_actionPerformed(java.awt.event.ActionEvent event) {
        setVisible(false);
        dispose();
    }

    // unused mouse events for current implementation
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
    public void mousePressed(MouseEvent e) {
    }
    public void mouseClicked(MouseEvent e) {
    }
    /**
     * standard mouse released event to get component list item
     */
    public void mouseReleased(java.awt.event.MouseEvent event) {
        Object object = event.getSource();
        if (object == componentList)
            componentList_mouseReleased(event);
    }

    /**
     * View all decomposed components of the selected component
     */
    void componentList_mouseReleased(java.awt.event.MouseEvent event) {
        if(event.getClickCount() > 0){
            decomposeListModel.removeAllElements();
            if (this.tf != null ){
                String currentComponent = this.tf.checkSelection((String)this.componentList.getSelectedValue());
            }
        }
    }

```

```

        if( !currentComponent.equals("") ){
            String currentPath = this.tf.convertToThePath(currentComponent);
            if( (currentPath != null) && (!currentPath.equals("")) ){
                File f = new File(currentPath);
                if( f.exists() ){
                    Vector decomposedSteps = new Vector();
                    decomposedSteps = (Vector)findSteps(currentComponent,new File(currentPath));
                    if( (decomposedSteps != null) && (decomposedSteps.size() > 0) ){
                        for( int i=0; i<decomposedSteps.size(); i++ ){
                            decomposeListModel.addElement(decomposedSteps.elementAt(i));
                        }
                    }
                }
            }
            else{
                JOptionPane.showMessageDialog(this, currentPath+" step is an atomic component. Please reselect it!",
                    "Warning Message",JOptionPane.ERROR_MESSAGE);
                return;
            }
        }
    }
}

/**
 * Find all decomposed components of the selected component
 *
 * @param temp : a string of selected component name
 * @param f : a file with a name temp
 * @return decomposes vector contains all decomposed components of the component temp
 */
Vector findSteps(String temp, File f) {
    Vector decomposes = new Vector();
    String[] list = (String[])f.list();
    for(int i=0; i<list.length; i++ ){
        File f2 = new File(f, list[i]);
        if( f2.isDirectory() ){
            try{
                int num = Integer.parseInt(f2.getName());
                int index = temp.indexOf("-");
                String temp2 = null;
                if( index > 0 ){
                    temp2 = temp+"."+f2.getName();
                }
                else{
                    temp2 = temp+"-"+f2.getName();
                }
                decomposes.addElement(temp2);
                findSteps(temp2,f2);
            }
            catch(Exception e){}
        }
    }
    return decomposes;
}
}

```

7. Cases.DeleteDialog

```

/**-----
 * Filename: DeleteDialog.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: JDK 1.3.1
 * Description: a dialog uses to delete a personnel object and a project

```

```

* Modified with standard events and listeners
*-----
**/
package Cases;

import java.awt.*;

import java.awt.event.ActionListener;

import java.awt.event.ItemListener;

import java.io.File;

import javax.swing.*;

////////////////////////////////////
/**
 * DeleteDialog : a dialog uses to delete a personnel object and
 * a project
 */
////////////////////////////////////
public class DeleteDialog extends javax.swing.JDialog implements ActionListener, ItemListener
{
    /**
     * fileDialog : use to get a personnel objects under stakeholder directory
     */
    FileDialog fileDialog = null;

    /**
     * deleteProject : a flag to define a purpose of using this dialog
     */
    boolean deleteProject = false;

    /**
     * casesFrame : a parent frame launch this dialog from two menu items,
     * Project --> Delete Project and
     * Tools --> Personnel --> Delete
     */
    CasesFrame casesFrame = null;

    /**
     * casesDirectory : the entire path of current project, eg. F:\Cases\projectName\
     */
    public String casesDirectory = null;

    /**
     *
     */
    public DeleteDialog(Frame parent)
    {
        super(parent);

        //{{ INIT_CONTROLS
        setTitle("Delete Project");
        setModal(true);
        getContentPane().setLayout(null);
        setSize(300,130);
        setVisible(false);
        OKButton.setText("OK");
        OKButton.setActionCommand("OK");
        getContentPane().add(OKButton);
        OKButton.setBounds(74,70,73,20);
        cancelButton.setText("Cancel");
        cancelButton.setActionCommand("Cancel");
        getContentPane().add(cancelButton);
        cancelButton.setBounds(152,70,73,20);
        //}}
    }
}

```

```

        //{{REGISTER_LISTENERS

        OKButton.addActionListener(this);
        cancelButton.addActionListener(this);
        browseButton.addActionListener(this);

        projectComboBox.addItemListener(this);
        //}}

    }

/**
 * default constructor
 */
    public DeleteDialog() {
        this((Frame)null);
    }

/**
 * CasesFrame uses this constructor to launch this dialog
 *
 * @param casesFrame : current CasesFrame
 * @param title : "Delete Project" or "Delete Personnel Data"
 */
    public DeleteDialog(CasesFrame casesFrame, String title){
        this(title);
        this.casesFrame = casesFrame;
        if( title.equals("Delete Project") ){
            getContentPane().add(projectComboBox);
            projectComboBox.setBounds(161,30,128,20);
            projectTextField.setEditable(false);
            getContentPane().add(projectTextField);
            projectTextField.setBackground(java.awt.Color.white);
            projectTextField.setBounds(10,30,150,20);

            casesDirectory = this.casesFrame.CASESDIRECTORY.getAbsolutePath() + "\\";
            projectTextField.setText(casesDirectory);

            String[] list = this.casesFrame.CASESDIRECTORY.list();

            if( list.length > 0 ){
                for(int i=0; i<list.length; i++){
                    File aFile = new File(casesDirectory, list[i]);
                    if( aFile.isDirectory() ){
                        projectComboBox.addItem(list[i]);
                    }
                }
            }
            deleteProject = true;
        }
        else if( title.equals("Delete Personnel Data") ){
            fileDialog = new FileDialog(this.casesFrame, title);
            getContentPane().add(filenameTextField);
            filenameTextField.setBounds(10,30,200,20);
            browseButton.setText("Browse");
            browseButton.setActionCommand("Browse");
            getContentPane().add(browseButton);
            browseButton.setBounds(211,30,78,20);

            deleteProject = false;
        }
    }

/**
 * polymorphic constructor to set title of dialog to sTitle
 * @param String
 */
    public DeleteDialog(String sTitle)    {

```

```

        this();
        setTitle(sTitle);
    }

/**
 * method to control the visiblilty of the dialog based upon b
 * @param boolean
 */
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

/**
 * a main procedure for unit testing
 */
    static public void main(String[] args) {
        (new DeleteDialog()).setVisible(true);
    }

/**
 * this method overrides the super addNotify() method
 */
    public void addNotify() {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets
        Insets insets = getInsets();
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    //{{DECLARE_CONTROLS
    javax.swing.JTextField filenameTextField = new javax.swing.JTextField();
    javax.swing.JButton OKButton = new javax.swing.JButton();
    javax.swing.JButton cancelButton = new javax.swing.JButton();
    javax.swing.JButton browseButton = new javax.swing.JButton();
    javax.swing.JComboBox projectComboBox = new javax.swing.JComboBox();
    javax.swing.JTextField projectTextField = new javax.swing.JTextField();
    //}}

/**
 * handles standard action events
 */
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == OKButton) // okay button pressed
            OKButton_actionPerformed(event);
        else if (object == cancelButton) // cancel button pressed
            cancelButton_actionPerformed(event);
        else if (object == browseButton) // browse button pressed
            browseButton_actionPerformed(event);
    }

/**
 * To check deleteProject or deletePersonnel before delete it

```

```

*/
    void OKButton_actionPerformed(java.awt.event.ActionEvent event)    {
    if( deleteProject ){
        File aFile = new File(projectTextField.getText());
        removeAll(aFile);
    }
    else{
        File aFile = new File(filenameTextField.getText());
        aFile.delete();
    }
    setVisible(false);
    dispose();
    }

/**
 * Exit DeleteDialog
 */
    void cancelButton_actionPerformed(java.awt.event.ActionEvent event)    {
    setVisible(false);
    dispose();
    }

/**
 * Show fileDialog with current directory is stakeholder and list all personnel file
 */
    void browseButton_actionPerformed(java.awt.event.ActionEvent event)    {
    fileDialog.setMode(FileDialog.LOAD);
    fileDialog.setDirectory(this.casesFrame.STAKEHOLDER.getAbsolutePath());
    fileDialog.show();
    if( fileDialog.getFile() != null ){
        filenameTextField.setText(fileDialog.getDirectory()+fileDialog.getFile());
    }
    else{
        filenameTextField.setText("");
    }
    }

/**
 * method to handle item state changes for project combo box
 */
    public void itemStateChanged(java.awt.event.ItemEvent event)    {
        Object object = event.getSource();
        if (object == projectComboBox)
            projectComboBox_itemStateChanged(event);
    }

/**
 * projectComboBox contains all project names under Cases directory
 * a user allows to select a project which they want to delete
 */
    void projectComboBox_itemStateChanged(java.awt.event.ItemEvent event)    {
    if( event.getStateChange() == event.SELECTED ){
        String s = (String)event.getItem();

        projectTextField.setText(casesDirectory+s);
    }
    }

/**
 * Remove all files under the deleting project before delete the project folder
 *
 * @param aFile : the deleting project folder
 */
void removeAll(File aFile){
    String[] l = aFile.list();
    for( int i=0; i<l.length; i++ ){
        File file = new File(aFile, l[i]);

```

```

        if( file.isDirectory() ){
            removeAll(file);
        }
        else{
            file.delete();
        }
    }
    aFile.delete();
}
}

```

8. Cases.Dependency

```

/**-----
 * Filename: Dependency.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Compiler: JDK 1.3.1
 * Description: Create a Dependency object and save it in dependency.cfg file
 *-----
 */

package Cases;

/**
 * Dependency object which is used to save in the dependency.cfg file
 */

import java.io.Serializable;

////////////////////////////////////
/**
 * Dependency : Create a Dependency object and save it in dependency.cfg file
 */
////////////////////////////////////
public class Dependency implements Serializable {

    /**
     * loopName : evolution process name
     */
    private String loopName = null;

    /**
     * step : step type name
     */
    private String step = null;

    /**
     * outputComponent : output component of the step type
     */
    private String outputComponent = null;

    /**
     * primaryInput : primary input of the step type
     */
    private String primaryInput = null;

    /**
     * secondaryInput : secondary input of the step type
     */
    private String secondaryInput = null;;

    /**
     * This Dependency constructor is used to create a Dependency object
     */
    public Dependency( String loopName, String step, String outputComponent, String primaryInput, String secondaryInput ){
        this.loopName = loopName;

```



```

        this.step = step;
        this.outputComponent = outputComponent;
        this.primaryInput = primaryInput;
        this.secondaryInput = secondaryInput;
    }

    /** default constructor */
    public Dependency() {}

    /**
     * @return loopName : evolution process name
     */
    public String getLoopName(){
        return this.loopName;
    }

    /**
     * @return step : step type name
     */
    public String getStep(){
        return this.step;
    }

    /**
     * @return outputComponent : output component of the step type
     */
    public String getOutputComponent(){
        return this.outputComponent;
    }

    /**
     * @return primaryInput : primary input of the step type
     */
    public String getPrimaryInput(){
        return this.primaryInput;
    }

    /**
     * @return secondaryInput : secondary input of the step type
     */
    public String getSecondaryInput(){
        return this.secondaryInput;
    }

    /**
     * Set secondary input for this Dependency object
     *
     * @param secondaryInput : secondary input component
     */
    public void setSecondaryInput( String secondaryInput ){
        this.secondaryInput = secondaryInput;
    }
}

```

9. Cases.DependencyType

```

/**-----
 * Filename: DependencyType.java
 * @Date: 2-12-2003
 * @Author: Arthur Clomera for NPS Thesis
 * @Compiler: JDK 1.3.1
 * Description: Object to store QFD dependency data
 *-----
 */
package Cases;

import java.io.Serializable;

```

```

/**
 * DependencyType : Create a DependencyType object and save it in depAttrib.cfg file
 */
public class DependencyType implements Serializable {
    /** Default constructor */
    public DependencyType() {
        this.setNewDependency(true);
    }

    /** returns the dependency name */
    public String getName(){ return name; }
    /** sets the dependency name */
    public void setName(String name){ this.name = name; }
    /** returns the dependency description */
    public String getDescription(){ return description; }
    /** sets the dependency description */
    public void setDescription(String description){ this.description = description; }
    /** returns the dependency type */
    public int getType(){
        return type;
    }
    /** sets the dependency type */
    public void setType(int type){
        this.type = type;
    }
    /** return the value range of possible input */
    public int getValueRange(){
        return valueRange;
    }
    /** set the value range of possible input */
    public void setValueRange(int valueRange){
        this.valueRange = valueRange;
    }
    /** return the default value to initialize QFD house of quality */
    public String getDefaultValue(){ return defaultValue; }
    /** set the default value to initialize QFD house of quality */
    public void setDefaultValue(String defaultValue){ this.defaultValue = defaultValue; }
    /** return the origin */
    public String getOrigin(){
        return origin;
    }
    /** set the origin */
    public void setOrigin(String origin){
        this.origin = origin;
    }
    /** return boolean value to question Is this a new dependency? (never used) */
    public boolean isNewDependency(){ return newDependency; }
    /** set boolean value to question Is this a new dependency? */
    public void setNewDependency(boolean newDependency){ this.newDependency = newDependency; }
    /** returns string value of origin version */
    public String getDepVersion(){ return depVersion; }
    /** set string value of origin version */
    public void setDepVersion(String depVersion){ this.depVersion = depVersion; }

    /** dependency name */
    private String name;
    /** dependency description */
    private String description;
    /** dependency type: risk, safety, parent-child */
    private int type;
    /** dependency value range 0:0..9, 1:true/false, etc... */
    private int valueRange;
    /** dependency default value (stored as a string, but should be a number */
    private String defaultValue;
    /** dependency origin (which component to base orientation of calculations on) */
    private String origin;
    /** dependency flag : true -> use default value false -> use data from object */
    private boolean newDependency;
    /** dependency version : stores the version of the dependency origin */

```

```

private String depVersion;
}

```

10. Cases.EHL

```

/**-----
 * Filename: EHL.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Compiler: JDK 1.3.1
 * Description: Create a EHL object and save it in loop.cfg file
 *-----
 */
package Cases;

import java.io.Serializable;

////////////////////////////////////
/**
 * EHL : Create a EHL object and save it in loop.cfg file
 */
////////////////////////////////////
public class EHL implements Serializable{

    /**
     * EHLName : evolution history process name
     */
    private String EHLName;

    /**
     * EHLPath : a string of step types in the evolution process
     */
    private String EHLPath;

    /**
     * This EHL constructor is used to create a EHL object
     */
    public EHL( String EHLName, String EHLPath ){
        if( EHLName == null ){
            EHLName = "";
        }

        if( EHLPath == null ){
            EHLPath = "";
        }
        this.EHLName = EHLName;
        this.EHLPath = EHLPath;
    }

    /**
     * default constructor
     */
    public EHL() {}

    /**
     * @return EHLName : evolution history process name
     */
    public String getEHLName(){
        return this.EHLName;
    }

    /**
     * @return EHLPath : a string of step types in the evolution process
     */
    public String getEHLPath(){
        return this.EHLPath;
    }
}

```

```
}
}
```

11. Cases.EditDecomposeFrame

```
/**-----
 * Filename: Ed8tDecomposeFrame.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: JDK 1.3.1
 * Description: use to edit/decompose a selected step
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_EditDecompose
 *-----
 */
package Cases;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.util.StringTokenizer;
import javax.swing.*;
import Cases.Interfaces.I_EditDecompose;

////////////////////////////////////
/**
 * EditDecomposeFrame : use to edit/decompose a selected step
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_EditDecompose
 */
////////////////////////////////////
public class EditDecomposeFrame extends javax.swing.JFrame implements CasesTitle, I_EditDecompose, ActionListener
{
    /**
     * pathName : an absolute path of a selected step
     */
    String pathName = null;

    /**
     * newPath : an absolute path after decompose a selected step
     */
    String newPath = null;

    /**
     * isEdit : a flag to keep track the purpose of using this frame
     */
    boolean isEdit = false;

    /**
     * Build EditDecomposeFrame
     */
    public EditDecomposeFrame() {
        initGUI();
    }

    /**
     * method to initialize GUI and its components
     */
    private void initGUI() {
        // This code is modified by Arthur Clomera to remove automatic code generated
        // by Visual Cafe
        //{ {INIT_CONTROLS
```

```

setTitle("Edit");
getContentPane().setLayout(null);
setSize(470,305);
setVisible(false);
titleLabel.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
titleLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
titleLabel.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
titleLabel.setText("TESTING: ");
getContentPane().add(titleLabel);
titleLabel.setForeground(java.awt.Color.black);
titleLabel.setFont(new Font("Dialog", Font.BOLD, 18));
titleLabel.setBounds(3,6,120,30);
JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel1.setText("Step Version");
getContentPane().add(JLabel1);
JLabel1.setForeground(java.awt.Color.black);
JLabel1.setBounds(7,54,175,22);
stepTextField.setEditable(false);
getContentPane().add(stepTextField);
stepTextField.setBackground(java.awt.Color.white);
stepTextField.setBounds(187,54,270,22);
JLabel2.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel2.setText("Output Component");
getContentPane().add(JLabel2);
JLabel2.setForeground(java.awt.Color.black);
JLabel2.setBounds(7,96,175,22);
outputTextField.setEditable(false);
getContentPane().add(outputTextField);
outputTextField.setBackground(java.awt.Color.white);
outputTextField.setBounds(187,96,270,22);
JLabel3.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel3.setText("Primary Input Component(s)");
getContentPane().add(JLabel3);
JLabel3.setForeground(java.awt.Color.black);
JLabel3.setBounds(7,138,175,22);
getContentPane().add(primaryTextField);
primaryTextField.setBounds(187,138,270,22);
JLabel4.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel4.setText("Secondary Input Component(s)");
getContentPane().add(JLabel4);
JLabel4.setForeground(java.awt.Color.black);
JLabel4.setBounds(7,180,175,22);
getContentPane().add(secondaryTextField);
secondaryTextField.setBounds(187,180,270,22);
selectedLabel.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
selectedLabel.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
selectedLabel.setText("Selected Label");
getContentPane().add(selectedLabel);
selectedLabel.setBackground(new java.awt.Color(204,204,204));
selectedLabel.setForeground(java.awt.Color.black);
selectedLabel.setFont(new Font("Dialog", Font.BOLD, 16));
selectedLabel.setBounds(130,6,290,30);
saveButton.setText("Save");
saveButton.setActionCommand("Save");
getContentPane().add(saveButton);
saveButton.setBounds(12,258,75,22);
exitButton.setText("Cancel");
exitButton.setActionCommand("Save");
getContentPane().add(exitButton);
exitButton.setBounds(372,258,75,22);
getContentPane().add(deleteButton);
deleteButton.setBounds(0,0,0,0);
//}}

//{{INIT_MENU
//}}

//{{REGISTER_LISTENERS

```

```

        saveButton.addActionListener(this);
        deleteButton.addActionListener(this);
        exitButton.addActionListener(this);
        //}}
    }
}

/**
 * CasesFrame launch this frame through SPIDER --> Edit
 */
public EditDecomposeFrame(String sTitle, String stepName,
    String output, String pathName) {
    this();
    //Set Delete button for Edit frame
    deleteButton.setText("Delete");
    deleteButton.setActionCommand("Delete");
    getContentPane().add(deleteButton);
    deleteButton.setBounds(84,258,75,22);

    setTitle(sTitle);
    this.titleLabel.setText( "Edit : " );
    this.selectedLabel.setText( pathName );
    this.stepTextField.setText( stepName );
    this.outputTextField.setText( output );
    this.pathName = pathName;

    try {
        File f = new File(pathName+"\\input.p");
        if( f.exists() ){
            DataInputStream primIn = new DataInputStream( new FileInputStream(f));
            if( primIn != null ){
                this.primaryTextField.setText( (String)primIn.readLine() );
            }
        }
        f = new File(pathName+"\\input.s");
        if( f.exists() ){
            DataInputStream secondIn = new DataInputStream( new FileInputStream(f));
            if( secondIn != null ){
                this.secondaryTextField.setText( (String)secondIn.readLine() );
            }
        }
    } catch( IOException io ){ System.out.println(io);}

    newPath = this.pathName;
    isEdit = true;
}

/**
 * CasesFrame launch this frame through SPIDER --> Decompose
 */
public EditDecomposeFrame(String sTitle, String stepName,
    String output, String pathName, int subDir) {
    this();

    setTitle(sTitle);
    this.titleLabel.setText( "Decompose:" );
    this.selectedLabel.setText( pathName );
    this.stepTextField.setText( stepName );
    this.outputTextField.setText( output );
    this.pathName = pathName;

    try {
        File f = new File(pathName+"\\input.p");
        if( f.exists() ){
            DataInputStream primIn = new DataInputStream( new FileInputStream(f));
            if( primIn != null ){
                this.primaryTextField.setText( (String)primIn.readLine() );
            }
        }
    }

```

```

    }
    f = new File(pathName+"\\input.s");
    if( f.exists() ){
        DataInputStream secondIn = new DataInputStream( new FileInputStream(f));
        if( secondIn != null ){
            this.secondaryTextField.setText( (String)secondIn.readLine() );
        }
    }
    catch( IOException io ){ System.out.println(io);}

    newPath = this.pathName + "\\ " + subDir;
}

/**
 * method to control the visibility of the frame based upon b
 * @param boolean
 */
public void setVisible(boolean b) {
    if (b)
        setLocation(50, 50);
    super.setVisible(b);
}

/**
 * main procedure for unit testing
 */
static public void main(String[] args) {
    (new EditDecomposeFrame()).setVisible(true);
}

/**
 * overrides super method addNotify()
 */
public void addNotify() {
    // Record the size of the window prior to calling parents addNotify.
    Dimension size = getSize();

    super.addNotify();

    if (frameSizeAdjusted)
        return;
    frameSizeAdjusted = true;

    // Adjust size of frame according to the insets and menu bar
    Insets insets = getInsets();
    javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
    int menuBarHeight = 0;
    if (menuBar != null)
        menuBarHeight = menuBar.getPreferredSize().height;
    setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
}

// Used by addNotify
boolean frameSizeAdjusted = false;

//{{DECLARE_CONTROLS
javax.swing.JLabel titleLabel = new javax.swing.JLabel();
javax.swing.JLabel JLabel1 = new javax.swing.JLabel();
javax.swing.JTextField stepTextField = new javax.swing.JTextField();
javax.swing.JLabel JLabel2 = new javax.swing.JLabel();
javax.swing.JTextField outputTextField = new javax.swing.JTextField();
javax.swing.JLabel JLabel3 = new javax.swing.JLabel();
javax.swing.JTextField primaryTextField = new javax.swing.JTextField();
javax.swing.JLabel JLabel4 = new javax.swing.JLabel();
javax.swing.JTextField secondaryTextField = new javax.swing.JTextField();
javax.swing.JLabel selectedLabel = new javax.swing.JLabel();
javax.swing.JButton saveButton = new javax.swing.JButton();

```

```

        javax.swing.JButton exitButton = new javax.swing.JButton();
        javax.swing.JButton deleteButton = new javax.swing.JButton();
    //}}

    //{{DECLARE_MENUS
    //}}

    /**
    * method to handle standard action events
    */
    public void actionPerformed(ActionEvent event) {
        Object object = event.getSource();
        if (object == saveButton) // save button pressed
            saveButton_actionPerformed(event);
        else if (object == deleteButton) // delete button pressed
            deleteButton_actionPerformed(event);
        else if (object == exitButton) // exit button pressed
            exitButton_actionPerformed(event);
    }

    /**
    * Save all the creations or the changes after decomposing and editing
    */
    public void saveButton_actionPerformed(java.awt.event.ActionEvent event) {
        File dir = new File(newPath);
        if( !dir.exists() ){
            dir.mkdir();
        }

        if( dir.isDirectory() ){
            try{
                if( !isEdit ){
                    File oldFile = new File(this.pathName, COMPONENT_CONTENT_DIR);
                    File newFile = new File(newPath, COMPONENT_CONTENT_DIR);
                    newFile.mkdir();
                    if( oldFile.exists() && newFile.isDirectory() ){
                        copyFile(oldFile, newFile);
                    }
                }

                if( checkInputs(secondaryTextField.getText()) ) {
                    FileOutputStream fileOutput = new FileOutputStream(new File(dir, "\\input.p"));
                    DataOutputStream depPrimary = new DataOutputStream( fileOutput );
                    if( depPrimary != null ){
                        if( !primaryTextField.getText().equals("") ){
                            depPrimary.writeBytes(primaryTextField.getText());
                        }
                    }
                }
                depPrimary.flush();
                depPrimary.close();
                fileOutput.close();

                fileOutput = new FileOutputStream(new File(dir, "\\input.s" ));
                DataOutputStream depSecondary = new DataOutputStream( fileOutput );
                if( depSecondary != null ){
                    if( !secondaryTextField.getText().equals("") ){
                        depSecondary.writeBytes(secondaryTextField.getText());
                    }
                }
                depSecondary.flush();
                depSecondary.close();
                fileOutput.close();
                exitButton_actionPerformed(null);
            }
        }
    }
    catch(FileNotFoundException fex ){ System.out.println(fex);}
    catch(IOException e){System.out.println(e);}
}
}

```



```

/**
 * Delete this selected step
 */
void deleteButton_actionPerformed(java.awt.event.ActionEvent event) {
    int result = JOptionPane.showConfirmDialog( this, "All the files and subdirectories will be deleted! Would you like to
continue?",
        "Confirm Message",JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
    //result = 0 ==> yes
    //result = 1 ==> no
    if( result == 0 ){
        File aFile = new File(this.pathName);
        if( aFile.exists() ){
            removeAll(aFile);
        }
    }
    exitButton_actionPerformed(null);
}

/**
 * Exit EditDecomposeFrame
 */
public void exitButton_actionPerformed(java.awt.event.ActionEvent event) {
    this.setEnabled(false);
    dispose();
}

/**
 * Copy all link files under Component Content directory into the new decomposed step
 *
 * @param oldFile : old link files of a parent file
 * @param newFile : new link files of new atomic
 */
public void copyFile( File oldFile, File newFile ){
    try{
        for( int i=0; i< LINK_FILE_NAMES.length; i++){
            File oldLink = new File(oldFile, LINK_FILE_NAMES[i]);
            File newLink = new File(newFile, LINK_FILE_NAMES[i]);
            FileWriter fileWriter = new FileWriter( newLink);
            BufferedWriter bw = new BufferedWriter( fileWriter );
            if( bw != null ){
                String line = null;
                FileReader fileReader = new FileReader(oldLink);
                BufferedReader br = new BufferedReader( fileReader);
                if( br != null ){
                    while( (line=br.readLine()) != null ){
                        bw.write(line);
                    }
                }
                br.close();
                fileReader.close();
            }
            bw.flush();
            bw.close();
            fileWriter.close();
        }
    }
    catch( FileNotFoundException f ){System.out.println(f);}
    catch( IOException io ){System.out.println(io);}
}

/**
 * Check all components in secondary textfield are valid or not
 *
 * @param secondary : a string of secondaryTextField
 * @return boolean : valid or invalide input
 */
public boolean checkInputs(String secondary ){

```

```

        String output = outputTextField.getText();
        String sub2 = null;

        if( secondary != null ){
            StringTokenizer st = new StringTokenizer(secondary, ",");
            while( st.hasMoreTokens() ){
                String s = (String)st.nextToken();
                int j = s.indexOf("-");
                if( j>0){
                    sub2 = (s.substring(j+1)).trim();
                    if( !sub2.equals(output) ){
                        char c = (char)sub2.charAt(0);
                        boolean isLetter = (new Character(c)).isLetter(c);
                        if( isLetter ){
                            JOptionPane.showMessageDialog(this, s+" is invalid component!",
                                "Error Message",JOptionPane.ERROR_MESSAGE);
                        }
                        return false;
                    }
                }
            }
        }
        return true;
    }

    /**
     * Remove all files under the deleting the selected step before delete the step folder
     *
     * @param aFile : the deleting the selected step folder
     */
    void removeAll(File aFile){
        String[] l = aFile.list();
        for( int i=0; i<l.length; i++){
            File file = new File(aFile, l[i]);
            if( file.isDirectory() ){
                removeAll(file);
            }
            else{
                file.delete();
            }
        }
        aFile.delete();
    }
}

```

12. Cases.JobScheduleMenu

```

/**
 * Filename: JobScheduleMenu.java
 * @since 10-29-2002
 * @author Hahn Le
 * @Modified by Arthur B. Clomera
 * Compiler: JDK 1.3.1
 * Description: This class is responsible for job scheduling.
 * Modified with standard action events and listeners
 */

```

```

package Cases;

```

```

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Vector;
import javax.swing.*;
import Cases.JobSchedule.JDialog_message;
import Cases.JobSchedule.JDialog_message1;
import Cases.JobSchedule.JFrame_assignjob;

```

```

import Cases.JobSchedule.JFrame_manage;
public class JobScheduleMenu extends JMenu implements ActionListener {
    JMenuItem jobManagementMenuItem = new JMenuItem("Scheduling");
    JMenuItem jobAssignMenuItem = new JMenuItem("Assignment");

    /**
     * The main window which owns this menu.
     */
    protected CasesFrame ownerWindow;

    public JobScheduleMenu(CasesFrame owner) {
        super("Job Schedule");
        this.setActionCommand("Tools");
        this.setMnemonic((int)'J');
        ownerWindow = owner;

        jobManagementMenuItem.setActionCommand("Scheduling");
        jobManagementMenuItem.setMnemonic((int)'S');
        this.add(jobManagementMenuItem);
        jobManagementMenuItem.addActionListener(this);

        jobAssignMenuItem.setActionCommand("Assignment");
        jobAssignMenuItem.setMnemonic((int)'A');
        this.add(jobAssignMenuItem);
        jobAssignMenuItem.addActionListener(this);
    }
    /**
     * this method handles standard action events
     */
    public void actionPerformed(ActionEvent event) {
        Object object = event.getSource();
        if (object == jobManagementMenuItem)
            jobManagementMenuItem_actionPerformed(event);
        else if (object == jobAssignMenuItem)
            jobAssignMenuItem_actionPerformed(event);
    }
    ////////////////////////////////// JOB SCHEDULE //////////////////////////////////
    /**
     * Job Management
     */
    void jobManagementMenuItem_actionPerformed(ActionEvent event)
    {
        ownerWindow.person_queue = (Vector)ownerWindow.getPersonnelVector();
        ownerWindow.job_pool = (Vector)ownerWindow.getStepContentVector();
        for(int i=0; i<ownerWindow.job_pool.size(); i++){
            StepContent step=(StepContent)ownerWindow.job_pool.elementAt(i);
        }
        Predecessor();

        ownerWindow.cleanperson_job();//each time will kill all of person's job
        ownerWindow.checking_person();

        if(ownerWindow.job_queue.size()>0){
            (new JFrame_manage(ownerWindow.job_queue)).setVisible(true);
        }
        else{
            (new JDialog_message()).setVisible(true);
        }
    }

    /**
     * Job Assignment
     */
    void jobAssignMenuItem_actionPerformed(ActionEvent event)
    {

```

```

        if(ownerWindow.person_queue.size()>0 && ownerWindow.job_queue.size()>0){
            (new JFrame_assignjob(ownerWindow, ownerWindow.person_queue, ownerWindow.job_queue,
ownerWindow.job_pool)).setVisible(true);
        }
        else{
            (new JDialog_message1()).setVisible(true);
        }
    }
}

void Predecessor(){
    ownerWindow.job_queue.removeAllElements();
    ownerWindow.job_queue1.removeAllElements();
    ownerWindow.job_queue2.removeAllElements();
    for(int i=0; i<ownerWindow.job_pool.size(); i++){
        StepContent step=(StepContent)ownerWindow.job_pool.elementAt(i);
        if(step.getStatus().equals("Approved") ||step.getStatus().equals("Scheduled")){
            ownerWindow.job_queue1.addElement(step);
        }
    }
    for(int i=0; i<ownerWindow.job_queue1.size(); i++){
        StepContent step=(StepContent) ownerWindow.job_queue1.elementAt(i);
        step.setStatus("Scheduled");
        ownerWindow.setstep(step.getStepName(), step);

        Vector v=(Vector) step.getPredecessors();

        if(v!=null){
            int n=0;
            n=checkpredecesor(step);
            if(n>0){
                ownerWindow.job_queue.addElement(step);
            }
            else{
                ownerWindow.job_queue2.addElement(step);
            }
        }
        else{
            ownerWindow.job_queue.addElement(step);
        }
    }
}

int checkpredecesor(StepContent step){
    Vector v=(Vector) step.getPredecessors();
    if(v.size()>0){
        for(int i=0; i<v.size(); i++){
            String predecesor=(String) v.elementAt(i);
            for(int j=0; j<ownerWindow.job_queue1.size(); j++){
                StepContent step2=(StepContent) ownerWindow.job_queue1.elementAt(j);
                if(predecesor.equals(step2.getStepName())){
                    return 0;
                }
            }
        }
    }
    return 1;
}
}

```

13. Cases.ListDialog

```
/**
 * Filename: ListDialog.java
 * @since 10-29-2002
 * @author Hahn Le
 * @Modified by Arthur B. Clomera
 * Compiler: JDK 1.3.1 removed dependencies on Visual Cafe
 * Description: Use to list components. It is launched by StepContentFrame,
 *             TraceFrame, and ProjectSchemaFrame.
 *
 * Implement CasesTitle where stores all global variables of Cases package.
 * Modified with GUI and QFD subsystems and standard action events and listeners
 */
package Cases;

import java.awt.*;

import java.awt.event.ActionListener;

import java.io.File;

import java.util.Vector;

import javax.swing.*;

import Cases.GUI.GraphPanel;

import Cases.QFD.util.ExcelCSVLexer;

////////////////////////////////////
/**
 * ListDialog : Use to list components. It is launched by StepContentFrame,
 *             TraceFrame, and ProjectSchemaFrame.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 */
////////////////////////////////////
public class ListDialog extends javax.swing.JDialog implements CasesTitle, ActionListener
{
    /**
     * tf : trace frame, one of owner frames
     */
    TraceFrame tf = null;

    /**
     * scf : step component frame, one of owner frames
     */
    StepContentFrame scf = null;

    /**
     * csv : excel csv lexer, one of owner frames
     */
    ExcelCSVLexer csv = null;

    /**
     * gp : graph panel, one of owner frames
     */
    GraphPanel gp = null;

    /**
     * psf : project schema frame, one of owner frames
     */
    ProjectSchemaFrame psf = null;

    /**
     * listModel : model of the itemList
     */
    DefaultListModel listModel = new DefaultListModel();
}
```

```

/**
 * index : index of component content or trace button from TraceFrame
 */
int index = 0;

/**
 * Buil ListDialog
 */
public ListDialog(JFrame parent)
{
    super(parent);

    //{{INIT_CONTROLS
    setModal(true);
    getContentPane().setLayout(null);
    setSize(300,400);
    setVisible(false);
    getContentPane().add(itemScrollPane);
    itemScrollPane.setBounds(15,70,270,250);
    itemScrollPane.getViewport().add(itemList);
    itemList.setBounds(0,0,267,247);
    OKButton.setText("OK");
    getContentPane().add(OKButton);
    OKButton.setBounds(64,340,75,24);
    cancelButton.setText("Cancel");
    getContentPane().add(cancelButton);
    cancelButton.setBounds(160,340,75,24);
    titleLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
    titleLabel.setText("Jlabel");
    getContentPane().add(titleLabel);
    titleLabel.setForeground(java.awt.Color.black);
    titleLabel.setFont(new Font("Dialog", Font.BOLD, 18));
    titleLabel.setBounds(1,10,298,30);
    //}}

    //{{REGISTER_LISTENERS

    OKButton.addActionListener(this);
    cancelButton.addActionListener(this);
    //}}

    /**
     * Set model for the itemList
     */
    itemList.setModel(listModel);
}

/**
 * StepContentFrame launches this dialog with its components list
 *
 * @param scf : step content frame, owner frame
 * @param title : title of this dialog when step content frame launches it
 * @param itemVector : list of components from step content frame
 */
public ListDialog(StepContentFrame scf, String title, Vector itemVector)
{
    this((JFrame)scf);
    this.scf = scf;
    setTitle(title);
    this.titleLabel.setText(title);
    for( int i=0; i<itemVector.size(); i++ ){
        listModel.addElement(itemVector.elementAt(i));
    }
}

/**
 * StepContentFrame launches this dialog with its components list
 *

```

```

* @param scf : ExcelCSVLexer, owner frame
* @param title : title of this dialog when step content frame launches it
* @param itemVector : list of column headers
*/
    public ListDialog(ExcelCSVLexer csv, String title, Vector itemVector)
    {
        this((JFrame)null);
        this.csv = csv;
        setTitle(title);
        this.titleLabel.setText(title);
        for( int i=0; i<itemVector.size(); i++ ){
            listModel.addElement(itemVector.elementAt(i));
        }
    }

/**
 * StepContentFrame launches this dialog with its components list
 *
 * @param scf : GraphPanel, owner frame
 * @param title : title of this dialog when step content frame launches it
 * @param itemVector : list of column headers
 */
    public ListDialog(GraphPanel gp, String title, Vector itemVector)
    {
        this((JFrame)null);
        this.gp = gp;
        setTitle(title);
        this.titleLabel.setText(title);
        for( int i=0; i<itemVector.size(); i++ ){
            listModel.addElement(itemVector.elementAt(i));
        }
    }

/**
 * TraceFrame launches this dialog with its components list
 *
 * @param tf : trace frame, owner frame
 * @param title : title of this dialog when trace frame launches it
 * @param itemVector : list of components from trace frame
 * @param index : index of button from trace frame, eg, 0:trace button, or
 *                1:component content button
 */
    public ListDialog(TraceFrame tf, String title, Vector itemVector, int index)
    {
        this((JFrame)tf);
        this.tf = tf;
        this.index = index;
        setTitle(title);
        this.titleLabel.setText(title);
        for( int i=0; i<itemVector.size(); i++ ){
            listModel.addElement(itemVector.elementAt(i));
        }
    }

/**
 * ProjectSchemaFrame launches this dialog with its components list
 *
 * @param psf : project schema frame, owner frame
 * @param title : title of this dialog when project schema frame launches it
 * @param itemVector : list of components from project schema frame
 */
    public ListDialog(ProjectSchemaFrame psf, String title, Vector itemVector) {
        this((JFrame)psf);
        this.psf = psf;
        setTitle(title);
        this.titleLabel.setText(title);
        for( int i=0; i<itemVector.size(); i++ ){
            listModel.addElement(((ComponentType)itemVector.elementAt(i)).getComponentID());
        }
    }

```

```

/**
 * a method to control the visibility of dialog based upon b
 * @param boolean
 */
    public void setVisible(boolean b) {
        if (b){
            this.setLocationRelativeTo(this.scf);
        }
        super.setVisible(b);
    }

/**
 * overrides the super addNotify method
 */
    public void addNotify() {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets
        Insets insets = getInsets();
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    //{{DECLARE_CONTROLS
    javax.swing.JScrollPane itemScrollPane = new javax.swing.JScrollPane();
    javax.swing.JList itemList = new javax.swing.JList();
    javax.swing.JButton OKButton = new javax.swing.JButton();
    javax.swing.JButton cancelButton = new javax.swing.JButton();
    javax.swing.JLabel titleLabel = new javax.swing.JLabel();
    //}}

/**
 * method to handle standard action events
 */
    public void actionPerformed(java.awt.event.ActionEvent event) {
        Object object = event.getSource();
        if (object == OKButton) // okay button pressed
            OKButton_actionPerformed(event);
        else if (object == cancelButton) // cancel button pressed
            cancelButton_actionPerformed(event);
    }

/**
 * Return selected item(s) to the owner frame
 */
    void OKButton_actionPerformed(java.awt.event.ActionEvent event) {
        if (this.scf != null ){
            this.scf.setPredecessorComboBox(this.itemList.getSelectedValues());
            setVisible(false);
            dispose();
        }
        else if (this.csv != null) {
            this.csv.setColHeaderValue(this.itemList.getSelectedIndex());
            setVisible(false);
            dispose();
        }
        else if (this.tf != null ){

```



```

String selectedItem = (String)this.itemList.getSelectedValue();
if( index == 0 ){
    String s = this.tf.convertToThePath(selectedItem);
    if( s!= null ){
        this.tf.setSelectedItem(s, selectedItem);
        setVisible(false);
        dispose();
    }
}
else if( index == 1 ){
    String currentComponent = (String)this.tf.checkSelection(selectedItem);
    if( !currentComponent.equals("") ){
        File f = (File)this.tf.searchFilePath(currentComponent);
        if( !f.exists() ){
            JOptionPane.showMessageDialog(this, selectedItem+" does not exist!", "Alert Message",
                JOptionPane.ERROR_MESSAGE);
        }
        else {
            String s = this.tf.convertToThePath(currentComponent);
            if( s!= null ){
                this.tf.setSelectedItem(s, selectedItem);
                setVisible(false);
                dispose();
                this.tf.setComponentContent(selectedItem, f);
            }
            setVisible(false);
            dispose();
        }
    }
}
else if( this.psf != null ){
    this.psf.setItemList(this.itemList.getSelectedValues());
    setVisible(false);
    dispose();
}
else if( this.gp != null ) {
    this.gp.setDeplItemList(this.itemList.getSelectedIndex());
//    System.out.println("ListDialog:depitemlist: "+this.itemList.getSelectedIndex());
    setVisible(false);
    dispose();
}
}

/**
 * Exit ListDialog
 */
void cancelButton_actionPerformed(java.awt.event.ActionEvent event) {
    setVisible(false);
    dispose();
}
}

```

14. Cases.Personnel

```

/**
 * Filename: Personnel.java
 * @since 10-29-2002
 * @author Hahn Le
 * Compiler: JDK 1.3.1
 * Description: Create a Personnel object and save it in a file with
 *              the name is Personnel's ID under stakeholder directory
 */
package Cases;

import java.io.Serializable;

```

```

import java.util.Vector;
/**
 * Personnel Data object which is used to save
 * under stakeholder directory
 */

////////////////////////////////////
/**
 * Personnel : Create a Personnel object and save it in a file with
 * the name is Personnel's ID under stakeholder directory
 */
////////////////////////////////////
public class Personnel implements Serializable{

    /**
     * ID : personnel ID
     */
    private String ID = null;

    /**
     * name : name of a person
     */
    private String name = null;

    /**
     * skill : vector of the person's skills
     */
    private Vector skill = new Vector();

    /**
     * securityLevel : security level of the person
     */
    private int securityLevel = 0;

    /**
     * email : e-mail address of the person
     */
    private String email = null;

    /**
     * telephone : telephone number of the person
     */
    private String telephone = null;

    /**
     * fax : fax number of the person
     */
    private String fax = null;

    /**
     * address : address of the person
     */
    private String address = null;

    /**
     * majorJobs : major job list for this person
     */
    private Vector majorJobs = new Vector();

    /**
     * minorJobs : minor job list for this person
     */
    private Vector minorJobs = new Vector();

    /**
     * This Personnel constructor is used to create a Personnel object
     */

```

```

public Personnel(String ID, String name, Vector skill,
                 int securityLevel, String email, String telephone,
                 String fax, String address){
    this();
    this.ID = ID;
    this.name = name;
    this.skill = skill;
    this.securityLevel = securityLevel;
    this.email = email;
    this.telephone = telephone;
    this.fax = fax;
    this.address = address;
}

/**
 * Empty Personnel constructor
 */
public Personnel(){
}

/**
 * Set ID for this personnel object
 *
 * @param s : string of ID
 */
public void setID(String s){
    this.ID = s;
}

/**
 * ID of this personnel object
 *
 * @return ID : this person's ID
 */
public String getID(){
    return this.ID;
}

/**
 * Set name for this personnel object
 *
 * @param s : this person's name
 */
public void setName(String s){
    this.name = s;
}

/**
 * The person's name
 *
 * @return name : the name of person
 */
public String getName(){
    return this.name;
}

/**
 * Set skills for the person
 *
 * @param v : list of person's skill
 */
public void setSkill(Vector v){
    this.skill = v;
}

/**
 * Skills of the person
 */

```

```

    * @return skill : a vector of skills
    */
    public Vector getSkill(){
        return this.skill;
    }

    /**
     * Set security level for the person
     *
     * @param i : security level
     */
    public void setSecurityLevel(int i){
        this.securityLevel = i;
    }

    /**
     * Security level of the person
     *
     * @return securityLevel : security level
     */
    public int getSecurityLevel(){
        return this.securityLevel;
    }

    /**
     * Set email for the person
     *
     * @param s : email address of the person
     */
    public void setEmail(String s){
        this.email = s;
    }

    /**
     * Email address of the person
     *
     * @return email : email address of the person
     */
    public String getEmail(){
        return this.email;
    }

    /**
     * Set telephone number of the person
     *
     * @param s : a string of telephone number
     */
    public void setTelephone(String s){
        this.telephone = s;
    }

    /**
     * Telephone number of the person
     *
     * @return telephone : a string of telephone number
     */
    public String getTelephone(){
        return this.telephone;
    }

    /**
     * Set fax number for the person
     *
     * @param s : a string of fax number
     */
    public void setFax(String s){
        this.fax = s;
    }

```

```

/**
 * Fax number of the person
 *
 * @return s : a string of fax number
 */
public String getFax(){
    return this.fax;
}

/**
 * Set address for the person
 *
 * @param s : a string of address
 */
public void setAddress(String s){
    this.address = s;
}

/**
 * Address of the person
 *
 * @return address : a string of address
 */
public String getAddress(){
    return this.address;
}

/**
 * Set the list of minor jobs for the person
 *
 * @param v : a list of minor jobs
 */
public void setMinorJobs(Vector v){
    this.minorJobs = v;
    for( int i=0; i<v.size(); i++){
        System.out.println(i+" : MINORJOB = "+v.elementAt(i));
    }
}

/**
 * The list of minor jobs for the person
 *
 * @return minorJobs : a vector of minor jobs
 */
public Vector getMinorJobs(){
    return this.minorJobs;
}

/**
 * Set the list of major jobs for the person
 *
 * @param v: a vector of major jobs
 */
public void setMajorJobs(Vector v){
    this.majorJobs = v;
    for( int i=0; i<v.size(); i++){
        System.out.println(i+" : MAJORJOB = "+v.elementAt(i));
    }
}

/**
 * The list of major jobs for the person
 *
 * @return majorJobs : a vector of major jobs
 */
public Vector getMajorJobs(){
    return this.majorJobs;
}

```

```
}
}
```

15. Cases.PersonnelFrame

```
/**
 * Filename: PersonnelFrame.java
 * @since 10-29-2002
 * @author Hahn Le
 * @Modified by Arthur B. Clomera
 * Compiler: JDK 1.3.1 removed dependencies on Visual Cafe
 * Description: to create, edit, and view personnel object
 */
package Cases;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import Cases.Interfaces.I_Personnel;
import Cases.GUI.util.Tools;

/**
 * PersonnelFrame : to create, edit, and view personnel object
 */
/** Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_Personnel
 */
public class PersonnelFrame extends JFrame implements CasesTitle, I_Personnel, ActionListener, ItemListener
{
    /**
     * selectedSkill : contains all selected skill from SkillTableFrame
     */
    public Vector selectedSkill = new Vector();

    /**
     * view : a flag to define the purpose of using this frame, eg. view or edit
     */
    boolean view = false;

    /**
     * edit : a flag to define the purpose of using this frame, eg. view or edit
     */
    boolean edit = false;

    Personnel personnel = null;

    /**
     * listHeadings : the 3 column headings of job multi list
     */
    String[] listHeadings = {"Job Name", "Real Start Time", "Estimated Duration"};
    private DefaultTableModel jobs = new DefaultTableModel(listHeadings, 1);

    /**
     * Build PersonnelFrame and CasesFframe launch this to create personnel object
     */
}
```

```

        public PersonnelFrame() {
            initGUI();
        }
    /**
     * method to initialize GUI and its components
     */
    private void initGUI() {
        //{{ INIT_CONTROLS
        setTitle("Personnel Data");
        getContentPane().setLayout(null);
        setVisible(false);
        setSize(500,540);
        JLabel2.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel2.setText("ID");
        getContentPane().add(JLabel2);
        JLabel2.setForeground(java.awt.Color.black);
        JLabel2.setBounds(45,40,110,24);
        JLabel5.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel5.setText("E-mail Address");
        getContentPane().add(JLabel5);
        JLabel5.setForeground(java.awt.Color.black);
        JLabel5.setBounds(45,180,110,24);
        JLabel6.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel6.setText("Fax Number");
        getContentPane().add(JLabel6);
        JLabel6.setForeground(java.awt.Color.black);
        JLabel6.setBounds(45,250,110,24);
        JLabel7.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel7.setText("Address");
        getContentPane().add(JLabel7);
        JLabel7.setForeground(java.awt.Color.black);
        JLabel7.setBounds(45,285,110,24);
        JLabel8.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel8.setText("Name");
        getContentPane().add(JLabel8);
        JLabel8.setForeground(java.awt.Color.black);
        JLabel8.setBounds(45,75,110,24);
        JLabel9.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel9.setText("Security Level");
        getContentPane().add(JLabel9);
        JLabel9.setForeground(java.awt.Color.black);
        JLabel9.setBounds(45,145,110,24);
        JLabel10.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel10.setText("Telephone Number");
        getContentPane().add(JLabel10);
        JLabel10.setForeground(java.awt.Color.black);
        JLabel10.setBounds(45,215,110,24);
        JLabel11.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        JLabel11.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        JLabel11.setText("Personnel Data");
        getContentPane().add(JLabel11);
        JLabel11.setForeground(java.awt.Color.black);
        JLabel11.setFont(new Font("Dialog", Font.BOLD, 20));
        JLabel11.setBounds(0,2,500,40);
        getContentPane().add(IDTextField);
        IDTextField.setBackground(java.awt.Color.white);
        IDTextField.setForeground(java.awt.Color.black);
        IDTextField.setBounds(155,40,300,24);
        getContentPane().add(nameTextField);
        nameTextField.setBackground(java.awt.Color.white);
        nameTextField.setForeground(java.awt.Color.black);
        nameTextField.setBounds(155,75,300,24);
        getContentPane().add(clearButton);
        clearButton.setBounds(0,0,0,0);
        skillButton.setText("Skill");
        skillButton.setActionCommand("Skill");
        getContentPane().add(skillButton);
        skillButton.setBounds(55,110,100,24);
    }
}

```

```

exitButton.setText("Exit");
exitButton.setActionCommand("Exit");
getContentPane().add(exitButton);
exitButton.setBounds(0,0,0,0);
getContentPane().add(saveButton);
saveButton.setBounds(0,0,0,0);
getContentPane().add(skillComboBox);
skillComboBox.setBounds(155,110,300,24);
getContentPane().add(securityComboBox);
securityComboBox.setBounds(155,145,300,24);
getContentPane().add(emailTextField);
emailTextField.setBackground(java.awt.Color.white);
emailTextField.setForeground(java.awt.Color.black);
emailTextField.setBounds(155,180,300,24);
getContentPane().add(telephoneTextField);
telephoneTextField.setBackground(java.awt.Color.white);
telephoneTextField.setForeground(java.awt.Color.black);
telephoneTextField.setBounds(155,215,300,24);
getContentPane().add(addressTextField);
addressTextField.setBackground(java.awt.Color.white);
addressTextField.setForeground(java.awt.Color.black);
addressTextField.setBounds(155,285,300,24);
getContentPane().add(faxTextField);
faxTextField.setBackground(java.awt.Color.white);
faxTextField.setForeground(java.awt.Color.black);
faxTextField.setBounds(155,250,300,24);
getContentPane().add(JLabel1);
JLabel1.setBounds(0,0,0,0);
getContentPane().add(jobsScrollPane);
jobsScrollPane.setBounds(0,0,0,0);
getContentPane().add(JLabel3);
JLabel3.setBounds(0,0,0,0);
getContentPane().add(jobScrollPane);
jobScrollPane.setBounds(0,0,0,0);
onHandsPanel.setLayout(new FlowLayout(FlowLayout.CENTER,5,5));
getContentPane().add(onHandsPanel);
onHandsPanel.setBounds(0,0,0,0);
getContentPane().add(minorRadioButton);
minorRadioButton.setBounds(0,0,0,0);
getContentPane().add(majorRadioButton);
majorRadioButton.setBounds(0,0,0,0);
//$$ etchedBorder1.move(0,541);
//}}

//{{INIT_MENU
//}}

//{{REGISTER_LISTENERS

skillButton.addActionListener(this);
clearButton.addActionListener(this);
saveButton.addActionListener(this);
exitButton.addActionListener(this);

minorRadioButton.addItemListener(this);
majorRadioButton.addItemListener(this);
//}}

//Set security level combobox from 0..5
for( int i=0; i<SECURITY_LEVEL; i++ ){
    securityComboBox.addItem(i+"");
}

setSize(500,380);
clearButton.setText("Clear");
clearButton.setActionCommand("Delete");
getContentPane().add(clearButton);

```



```

        clearButton.setBounds(24,330,73,24);
        saveButton.setText("Save");
        saveButton.setActionCommand("Save");
        getContentPane().add(saveButton);
        saveButton.setBounds(96,330,73,24);
        exitButton.setBounds(380,330,73,24);
    }

    /**
     * CasesFrame launch this to edit and view personnel object
     *
     * @param selectedFile : a selected file name
     * @param request : purpose of using this frame, eg. View or Edit
     */
    public PersonnelFrame(String selectedFile, String request){
        this();
        System.out.println(selectedFile);
        try{
            File aFile = new File(selectedFile);
            if( aFile.exists() ){
                FileInputStream fileInput = new FileInputStream(aFile);
                ObjectInputStream oi = new ObjectInputStream( fileInput );
                if( oi != null ){
                    personnel = (Personnel)oi.readObject();
                }
                oi.close();
                fileInput.close();
            }

            catch( IOException io ){
                System.out.println("IOException: "+io);
            }
            catch( ClassNotFoundException c ){
                System.out.println("ClassNotFoundException: "+c);
            }
        }
        if( request.equals("Edit") || request.equals("Query") ){
            view = true;
            setSize(500,540);
            clearButton.setBounds(0,0,0,0);
            saveButton.setBounds(0,0,0,0);

            onHandsPanel.setBorder(BorderFactory.createEtchedBorder());

            onHandsPanel.setLayout(null);
            getContentPane().add(onHandsPanel);
            onHandsPanel.setBounds(155,320,300,30);
            minorRadioButton.setText("Minor Jobs");
            minorRadioButton.setSelected(true);
            onHandsPanel.add(minorRadioButton);
            minorRadioButton.setBounds(170,3,130,24);
            majorRadioButton.setText("Major Jobs");
            onHandsPanel.add(majorRadioButton);
            majorRadioButton.setBounds(20,3,130,24);
            majorRadioButton.setSelected(true);

            JLabel3.setHorizontalAlignment(SwingConstants.RIGHT);
            JLabel3.setText("On-hand Jobs");
            getContentPane().add(JLabel3);
            JLabel3.setForeground(java.awt.Color.black);
            JLabel3.setBounds(45,320,110,27);
            getContentPane().add(jobScrollPane);
            jobScrollPane.setBounds(45,355,410,130);
            jobScrollPane.getViewport().add(jobMultiList);
            jobMultiList.setBackground(java.awt.Color.white);
            jobMultiList.setBounds(0,0,407,127);
            exitButton.setBounds(384,495,73,24);
        }
        jobScrollPane.setVerticalScrollBarPolicy(javax.swing.JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    }

```

```

        jobScrollPane.setHorizontalScrollBarPolicy(javax.swing.JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);

        }
        else if( request.equals("Edit") ){
            edit = true;
        }
        if( personnel != null ){
            setInitial(personnel);
        }
    }

    /**
     * constructor to set title to sTitle
     * @param String
     */
    public PersonnelFrame(String sTitle)    {
        this();
        setTitle(sTitle);
    }

    /**
     * method to control visibility of frame based upon b
     * @param boolean
     */
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

    /**
     * main procedure for unit testing
     */
    static public void main(String[] args)    {
        (new PersonnelFrame()).setVisible(true);
    }

    /**
     * overrides super addNotify() method
     */
    public void addNotify()    {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    //{{DECLARE_CONTROLS

    /** converted visual cafe multilist to JTable */
    JTable jobMultiList = new JTable(jobs);

```

```

javax.swing.JLabel JLabel2 = new javax.swing.JLabel();
javax.swing.JLabel JLabel5 = new javax.swing.JLabel();
javax.swing.JLabel JLabel6 = new javax.swing.JLabel();
javax.swing.JLabel JLabel7 = new javax.swing.JLabel();
javax.swing.JLabel JLabel8 = new javax.swing.JLabel();
javax.swing.JLabel JLabel9 = new javax.swing.JLabel();
javax.swing.JLabel JLabel10 = new javax.swing.JLabel();
javax.swing.JLabel JLabel11 = new javax.swing.JLabel();
javax.swing.JTextField IDTextField = new javax.swing.JTextField();
javax.swing.JTextField nameTextField = new javax.swing.JTextField();
javax.swing.JButton clearButton = new javax.swing.JButton();
javax.swing.JButton skillButton = new javax.swing.JButton();
javax.swing.JButton exitButton = new javax.swing.JButton();
javax.swing.JButton saveButton = new javax.swing.JButton();
javax.swing.JComboBox skillComboBox = new javax.swing.JComboBox();
javax.swing.JComboBox securityComboBox = new javax.swing.JComboBox();
javax.swing.JTextField emailTextField = new javax.swing.JTextField();
javax.swing.JTextField telephoneTextField = new javax.swing.JTextField();
javax.swing.JTextField addressTextField = new javax.swing.JTextField();
javax.swing.JTextField faxTextField = new javax.swing.JTextField();
javax.swing.JLabel JLabel1 = new javax.swing.JLabel();
javax.swing.JScrollPane jobsScrollPane = new JScrollPane();
javax.swing.JLabel JLabel3 = new JLabel();
javax.swing.JScrollPane jobScrollPane = new javax.swing.JScrollPane();
javax.swing.JPanel onHandsPanel = new javax.swing.JPanel();
javax.swing.JRadioButton minorRadioButton = new javax.swing.JRadioButton();
javax.swing.JRadioButton majorRadioButton = new javax.swing.JRadioButton();

/**
 * Add all selected skills from SkillTableFrame to skillComboBox
 *
 * @param v : a selected skill vector
 */
public void setSkillComboBox(Vector v){
    this.selectedSkill = new Vector();
    this.skillComboBox.removeAllItems();
    if( v.size() > 0 ){
        this.skillComboBox.addItem("ID : Name : Level");
        for( int i=0; i<v.size(); i++ ){
            this.skillComboBox.addItem(v.elementAt(i));
            this.selectedSkill.addElement(v.elementAt(i));
        }
    }
}

/**
 * provides a method for handling standard action events
 */
public void actionPerformed(ActionEvent event)    {
    Object object = event.getSource();
    if (object == skillButton) // skill button pressed
        skillButton_actionPerformed(event);
    else if (object == clearButton) // clear button pressed
        clearButton_actionPerformed(event);
    else if (object == saveButton) // save button pressed
        saveButton_actionPerformed(event);
    else if (object == exitButton) // exit button pressed
        exitButton_actionPerformed(event);
}

/**
 * Launch SkillTableFrame
 */
public void skillButton_actionPerformed(ActionEvent event)    {
    (new SkillTableFrame(this)).setVisible(true);
}

```

```

/**
 * Get a personnel object and save it under stakeholder
 */
void saveButton_actionPerformed(ActionEvent event) {
    Personnel personnel = (Personnel)getPersonnelData();
    if( personnel != null ){
        try{
            FileOutputStream fileOutput = new FileOutputStream(STAKEHOLDER+"\\"+IDTextField.getText());
            ObjectOutputStream oo = new ObjectOutputStream(fileOutput);
            if( oo != null ){
                oo.writeObject(personnel);
            }
            oo.flush();
            oo.close();
            fileOutput.close();
        }
        catch(IOException io){
            JOptionPane.showMessageDialog(this, "Sorry, saving is unsuccessful", "Error Message",
JOptionPane.ERROR_MESSAGE);
        }
        setVisible( false );
        dispose();
    }
}

/**
 * Refresh all the textfields and combo boxes
 */
void clearButton_actionPerformed(ActionEvent event) {
    //clear all fields
    IDTextField.setText("");
    nameTextField.setText("");
    skillComboBox.removeAllItems();
    securityComboBox.setSelectedIndex(0);
    emailTextField.setText("");
    telephoneTextField.setText("");
    addressTextField.setText("");
    faxTextField.setText("");
}

/**
 * Exit PersonnelFrame
 */
void exitButton_actionPerformed(ActionEvent event) {
    setVisible( false);
    dispose();
}

/**
 * Create and return a personnel object
 *
 * @return personnel object
 */
public Personnel getPersonnelData(){
    Personnel personnel = new Personnel();
    try{
        String ID = IDTextField.getText();
        String name = nameTextField.getText();
        int security = Integer.parseInt((String)securityComboBox.getSelectedItem());
        String email = emailTextField.getText();
        String tel = telephoneTextField.getText();
        String fax = faxTextField.getText();
        String address = addressTextField.getText();

        if( ID.equals("") ||
            name.equals("") ||
            selectedSkill.size() == 0 ||
            email.equals("") ||

```

```

        tel.equals("") ||
        fax.equals("") ||
        address.equals("") ){
JOptionPane.showMessageDialog(this, "You did not complete all the fields", "Error Message",
JOptionPane.ERROR_MESSAGE);
return null;
    }
    else{
        personnel.setID(ID);
        personnel.setName(name);
        personnel.setSkill(selectedSkill);
        personnel.setSecurityLevel(security);
        personnel.setEmail(email);
        personnel.setTelephone(tel);
        personnel.setAddress(address);
        personnel.setFax(fax);
    }
}
catch(Exception e){}
return personnel;
}

/**
 * Set initial frame when it is launched to view and edit
 *
 * @param personnel : a personnel object
 */
public void setInitial(Personnel personnel){
    if( personnel != null ){
        IDTextField.setText(personnel.getID());
        nameTextField.setText(personnel.getName());
        setSkillComboBox(personnel.getSkill());
        securityComboBox.setSelectedItem(""+personnel.getSecurityLevel());
        emailTextField.setText(personnel.getEmail());
        telephoneTextField.setText(personnel.getTelephone());
        addressTextField.setText(personnel.getAddress());
        faxTextField.setText(personnel.getFax());
    }
    if( view ){
        IDTextField.setEditable(false);
        nameTextField.setEditable(false);
        skillButton.setEnabled(false);
        securityComboBox.setEnabled(false);
        emailTextField.setEditable(false);
        telephoneTextField.setEditable(false);
        addressTextField.setEditable(false);
        faxTextField.setEditable(false);
    }
}

/**
 * a method to manage item state changes of radio buttons
 */
public void itemStateChanged(ItemEvent event)    {
    Object object = event.getSource();
    if (object == minorRadioButton)
        minorRadioButton_itemStateChanged(event);
    else if (object == majorRadioButton)
        majorRadioButton_itemStateChanged(event);
}

/**
 * View minor jobs of selected personnel object to the list
 */
public void minorRadioButton_itemStateChanged(ItemEvent event)    {
    // get some tools
    Tools newTools = new Tools();
    if( event.getStateChange() == event.SELECTED ){

```

```

        majorRadioButton.setSelected(false);

// jobs.clear();
    for (int i=0;i<jobs.getRowCount(); i++)
        jobs.removeRow(i);
    Vector v = new Vector();
    if( personnel != null ){
        v = (Vector)personnel.getMinorJobs();
        for( int j=0; j<v.size(); j++ ){
            StepContent sc = (StepContent)v.elementAt(j);
            System.out.println("stepNamme = "+sc.getStepName());
            String number = newTools.intToString(sc.getDuration());
            String[] temp = { sc.getStepName(), sc.getRealStartTime(), number };
            jobs.addRow(temp);
        }
    }
}

/**
 * View major jobs of selected personnel object to the list
 */
public void majorRadioButton_itemStateChanged(ItemEvent event) {
    // get some tools
    Tools newTools = new Tools();
    if( event.getStateChange() == event.SELECTED ){
        minorRadioButton.setSelected(false);
        //jobs.clear();
        for (int i=0;i<jobs.getRowCount(); i++)
            jobs.removeRow(i);
        Vector v = new Vector();
        if( personnel != null ){
            v = (Vector)personnel.getMajorJobs();
            for( int j=0; j<v.size(); j++ ){
                StepContent sc = (StepContent)v.elementAt(j);
                System.out.println("stepNamme = "+sc.getStepName());
                String number = newTools.intToString(sc.getDuration());
                String[] temp = { sc.getStepName(), sc.getRealStartTime(), number };
                jobs.addRow(temp);
            }
        }
    }
}
}

```

16. Cases.ProjectMenu

```

/**
 * Filename: ProjectMenu.java
 * @since 10-29-2002
 * @author Hahn Le
 * @Modified by Arthur B. Clomera
 * Compiler: JDK 1.3.1 removed dependencies on Visual Cafe
 * Description: a separate class just for the cases menu.
 *
 * Implement ActionListener.
 * Modified with GUI subsystems and standard action events and listeners
 */
package Cases;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

```

```

import javax.swing.*.*;

public class ProjectMenu extends JMenu implements ActionListener {

    JMenuItem createProjectMenuItem = new JMenuItem("Create Project");
    JMenuItem openProjectMenuItem = new JMenuItem("Open Project");
    JMenuItem deleteProjectMenuItem = new JMenuItem("Delete Project");

    /**
     * Initiates the 'Quit' event
     */
    private JMenuItem exitMenuItem = new JMenuItem ("Exit");

    /**
     * The main window which owns this menu.
     */
    protected CasesFrame ownerWindow;

    /**
     * Constructor for this class.
     *
     * @param owner The main window which has created this menu.
     */
    public ProjectMenu (CasesFrame owner)
    {
        super ("Project");
        this.setActionCommand("Hypergraph");
        this.setMnemonic((int)'P');

        ownerWindow = owner;
        createProjectMenuItem.setActionCommand("Create Project");
        createProjectMenuItem.setMnemonic((int)'C');
        add (createProjectMenuItem);
        openProjectMenuItem.setActionCommand("Open Project");
        openProjectMenuItem.setMnemonic((int)'O');
        add (openProjectMenuItem);
        deleteProjectMenuItem.setActionCommand("Delete Project");
        deleteProjectMenuItem.setMnemonic((int)'D');
        add (deleteProjectMenuItem);
        addSeparator ();
        exitMenuItem.setActionCommand("Exit");
        exitMenuItem.setMnemonic((int)'X');
        add (exitMenuItem);

        /**
         * Register the action listeners
         */
        createProjectMenuItem.addActionListener (this);
        openProjectMenuItem.addActionListener (this);
        deleteProjectMenuItem.addActionListener (this);

        exitMenuItem.addActionListener (this);
    }

    /**
     * Action event handler for the menu events.
     *
     * @param e The action event that is created by selecting
     * a menu item from this menu
     */
    public void actionPerformed(ActionEvent event) {
        Object object = event.getSource();
        if (object == createProjectMenuItem)
            createProjectMenuItem_actionPerformed(event);
        else if (object == openProjectMenuItem)
            openProjectMenuItem_actionPerformed(event);
        else if (object == deleteProjectMenuItem)

```

```

        deleteProjectMenuItem_actionPerformed(event);
    else if (object == exitMenuItem)
        System.exit(0);
}
/**
 * Connect to DeleteDialog, where allows a user to delete a project
 * and all existing project names are in the combo box.
 * Except, the current project can not be deleted since it is occupying.
 *
 * @param event : action event from Delete Project menu item
 */
void deleteProjectMenuItem_actionPerformed(ActionEvent event)
{
    ownerWindow.drawPanel.clear();
    ownerWindow.drawToolBar.setButtons(false);
    ownerWindow.drawToolBar.setProjectName("Project: ");
    (new DeleteDialog(ownerWindow, "Delete Project")).setVisible(true);
}

/**
 * Ask a user to enter the new project name.
 * Do not allow a duplicate project name.
 * If the name is not duplicate, create the new project and connect to
 * ProjectSchemaFrame directly.
 *
 * @param event : action event from Create Project menu item
 */
void createProjectMenuItem_actionPerformed(ActionEvent event)
{
    // initialize the drawPanel
    ownerWindow.drawPanel.clear();
    ownerWindow.drawToolBar.setButtons(true);
    ownerWindow.drawToolBar.setCalcButton(false);
    ownerWindow.drawToolBar.setProjectName("Project: ");
    ownerWindow.drawPanel.depAttrib.removeAllElements();
    ownerWindow.projectName = JOptionPane.showInputDialog( ownerWindow, "Project Name",
        "Create Project",JOptionPane.INFORMATION_MESSAGE);
    if( ownerWindow.projectName != null ){
        ownerWindow.drawToolBar.setProjectName("Project: "+ownerWindow.projectName);
        String[] theList = ownerWindow.CASESDIRECTORY.list();
        if( theList.length == 0 ){
            ownerWindow.setOpenDelete( true );
            File newFile = new File(ownerWindow.CASESDIRECTORY,ownerWindow.projectName);
            newFile.mkdir();

            if( newFile.isDirectory() ){
                ownerWindow.pathName = newFile.getAbsolutePath();

                ownerWindow.psfWindow = new ProjectSchemaFrame(ownerWindow);
                ownerWindow.psfWindow.setVisible(true);
                ownerWindow.menuSetEnabled(false);
            }
        }
        else if( theList.length>0 ){
            for( int i=0; i<theList.length; i++ ){

                // Checking the duplication before create the new project
                if( ownerWindow.projectName.equals(theList[i]) ){
                    i= theList.length;
                    JOptionPane.showMessageDialog(ownerWindow, ownerWindow.projectName+" has been created.",
                        "Error Message", JOptionPane.ERROR_MESSAGE);

                    return;
                }
            }
            else if( i==(theList.length-1) ){
                ownerWindow.setOpenDelete( true );
                File newFile = new File(ownerWindow.CASESDIRECTORY,ownerWindow.projectName);
                newFile.mkdir();
                if( newFile.isDirectory() ){

```



```

        ownerWindow.pathName = newFile.getAbsolutePath();
        ownerWindow.psfWindow= new ProjectSchemaFrame(ownerWindow);
        ownerWindow.psfWindow.setVisible(false);
        ownerWindow.menuSetEnabled(true);
    }
}
}
}
}

/**
 * Allow a user to select a project, then he has a choice
 * Yes : connect to ProjectSchemaFrame to edit step.cfg, component.cfg, loop.cfg,
 * or dependency.cfg file.
 * No : go straight to main menu (Automatic Version Control, SPIDER, Tools, or Job Schedule
 *
 * @param event : action event from Open Project menu item
 */
void openProjectMenuItem_actionPerformed(ActionEvent event)
{
    int returnValue=-1;

    // initialize the drawPanel
    ownerWindow.drawPanel.clear();
    ownerWindow.drawToolBar.setButtons(true);
    ownerWindow.drawPanel.depAttrib.removeAllElements();
    // remove previous dependencies and disable sync button
    ownerWindow.drawToolBar.setCalcButton(false);
    ownerWindow.drawPanel.depAttrib.removeAllElements();

    if( ownerWindow.projectName != null ){
        File newFile = new File(ownerWindow.CASESDIRECTORY,ownerWindow.projectName);
        newFile.mkdir();
        ownerWindow.fileChooser = new JFileChooser(newFile);
    }
    ownerWindow.fileChooser.setDialogTitle("Open Project");
    ownerWindow.fileChooser.setCurrentDirectory(ownerWindow.CASESDIRECTORY);
    ownerWindow.fileChooser.setSelectionMode(ownerWindow.fileChooser.DIRECTORIES_ONLY);
    repaint();

    returnValue = ownerWindow.fileChooser.showDialog(ownerWindow,"Open");
    if( returnValue == ownerWindow.fileChooser.APPROVE_OPTION ){
        File file = ownerWindow.fileChooser.getCurrentDirectory();
        ownerWindow.projectName = ownerWindow.fileChooser.getSelectedFile().getName();
        ownerWindow.pathName = file.getAbsolutePath()+"\\"+ownerWindow.projectName;
        ownerWindow.psfWindow=new ProjectSchemaFrame(ownerWindow);
        ownerWindow.drawPanel.numComponents = ownerWindow.psfWindow.compHashtable.size();
        ownerWindow.drawToolBar.setProjectName("Project: "+ownerWindow.projectName);
        ownerWindow.drawPanel.repaint();
        ownerWindow.menuSetEnabled(true);
    }
}

} // End of the class ProjectMenu

```

17. Cases.ProjectSchemaFrame

```

/**-----
 * @Filename: ProjectSchemaFrame.java
 * @Date: 3-7-2003
 * @Author: Le Hahn but modified by Arthur Clomera for NPS Thesis
 * reduced the size and complexity of the original class by extracting GUI tab panel

```

```

* into their own classes: StepTypePanel, ComponentTypePanel, EHLTypePanel and
* DependencyTypePanel. Additional objects include: ObjectFileOperations
* @Compiler: JDK 1.3.1
* -----
**/
package Cases;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.StringTokenizer;
import java.util.Vector;
import javax.swing.*;
import Cases.GUI.GraphPanel;
import Cases.GUI.psf.ComponentTypePanel;
import Cases.GUI.psf.DependencyTypePanel;
import Cases.GUI.psf.EHLTypePanel;
import Cases.GUI.psf.StepTypePanel;
import Cases.Interfaces.I_ProjectSchema;
import Cases.QFD.ComponentQFD;
import Cases.QFD.MyDefaultTableModel;
import Cases.QFD.StepQFD;
import Cases.QFD.util.ObjectFileOperations;

/**
 * ProjectSchemaFrame : allows a user to create step types, component types,
 * evolution history loop, and dependency which are stored in step.cfg,
 * component.cfg, loop.cfg, and dependency.cfg respectively
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_ProjectSchema
 */

public class ProjectSchemaFrame extends JFrame implements CasesTitle, I_ProjectSchema, ActionListener, MouseListener,
ItemListener
{
    /**
     * saveTab : flags of 4 tabs and to confirm that every tab should be saved before exit the frame
     */
    boolean[] saveTab = {true, true, true, true};

    /**
     * stepVector : contains all step type objects
     */
    public Vector stepVector = new Vector();

    /**
     * compVector : contains all component type objects
     */
    public Vector compVector = new Vector();

    /**
     * EHLVector : contains all EHL objects
     */
    public Vector EHLVector = new Vector();

    /**
     * stepHashtable : with the keys and the objects are step type IDs and step type objects
     */
    public Hashtable stepHashtable = new Hashtable();

    /**
     * compHashtable : with the keys and the objects are component type IDs and component type objects
     */
    public Hashtable compHashtable = new Hashtable();

```

```

/**
 * EHLHashtable : with the keys and the objects are EHL IDs and EHL objects
 */
public Hashtable EHLHashtable = new Hashtable();

/**
 * depenHashtable : with the keys and the objects are dependency IDs and dependency objects
 */
public Hashtable depenHashtable = new Hashtable();

public String projectName;

/**
 * testIndex : to keep track the secondary input of each dependency object to be set correctly
 */
public int testIndex = 0;

public JComboBox existedCompComboBox = new JComboBox();
public JComboBox existedStepComboBox = new JComboBox();
public JTabbedPane configManagTabbedPane = new JTabbedPane();

/**
 * selectedStepIndex : to make sure that getting the correct step type object
 */
public int selectedStepIndex;
public int stepIndexFrom;
public int stepIndexTo;

private CasesFrame ownerWindow;

/**
 * selectedCompIndex : to make sure that getting the correct component type object
 */
public int selectedCompIndex;
public int compIndex;
public Point compPosition;

/**
 * selectedEHLIndex : to make sure that getting the correct EHL object
 */
public int selectedEHLIndex;

/**
 * selectedDepenStepIndex : to make sure that getting the correct dependency object
 */
public int selectedDepenStepIndex;

/**
 * pathName : the path of the current project
 */
public String pathName = CASESDIRECTORY.getAbsolutePath();

/**
 * listModel : monitor all step types in EHL panel
 */
public DefaultListModel listModel = new DefaultListModel();

/**
 * This method initializes the GUI and its components
 * @param void
 */
public void initGUI() {
    setTitle("Project Schema");
    getContentPane().setLayout(null);
    setSize(600,450);
    setVisible(false);
    mainPanel.setLayout(new GridLayout(1,1,0,0));
}

```

```

        getContentPane().add(mainPanel);
        mainPanel.setBounds(10,0,580,360);
        mainPanel.add(configManagTabbedPane);
        configManagTabbedPane.setForeground(java.awt.Color.black);
        configManagTabbedPane.setBounds(0,0,580,360);
        configManagTabbedPane.add(stepTypePanel);
        configManagTabbedPane.add(componentTypePanel);
        configManagTabbedPane.add(EHLPanel);
        configManagTabbedPane.add(dependencyPanel);

        this.existedStepComboBox.setBounds(225,150,300,22);
        this.existedStepComboBox.setMaximumRowCount(testIndex);
        this.existedCompComboBox.setBounds(225,150,300,22);

        configManagTabbedPane.setSelectedIndex(0);
        configManagTabbedPane.setSelectedComponent(stepTypePanel);
        try {
            configManagTabbedPane.setTitleAt(0,"Step Type");
        }
        catch(ArrayIndexOutOfBoundsException e) {
            System.out.println(e);
        }

        configManagTabbedPane.setTitleAt(1,"Component Type");
        configManagTabbedPane.setTitleAt(2,"");// software evolution process
        configManagTabbedPane.setTitleAt(3,"");// primary and secondary inputs dependencies
        doneButton.setText("Finish");
        doneButton.setActionCommand("OK");
        getContentPane().add(doneButton);
        doneButton.setBounds(262,390,75,22);
        doneButton.hide();

        //{REGISTER_LISTENERS
        doneButton.addActionListener(this);
        existedStepComboBox.addItemListener(this);
        existedCompComboBox.addItemListener(this);
        //}

//    setEnabled(0, false);
//    setEnabled(1, false);
//    setEnabled(2, false);
//        setEnabled(3, false);

    }

/**
 * Is launched when a user wants to create or edit step.cfg, component.cfg, loop.cfg, or dependency.cfg
 *
 * @param selectedProject : current project name
 * @param pathName : the path of the current project
 */
public ProjectSchemaFrame( CasesFrame owner ){
    ownerWindow=owner;
    this.projectName=ownerWindow.projectName;
    initGUI();
    this.setProjectLabel(ownerWindow.projectName );
    this.readInputFiles(ownerWindow.drawPanel );
}
/**
 * Is launched when a user wants to create or edit step.cfg, component.cfg, loop.cfg, or dependency.cfg
 *
 * @param selectedProject : current project name
 * @param pathName : the name of the current project
 */
public ProjectSchemaFrame( String project ){
    initGUI();
    this.projectName = project;

```

```

        this.setProjectLabel(project);
        this.readInputFiles(project);
    }

    /**
     * controls the visibility of frame based upon b
     * @param boolean
     */
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

    /**
     * overrides super addNotify() method
     */
    public void addNotify() {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        JMenuBar menuBar = getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    /**{DECLARE_CONTROLS
     JPanel mainPanel = new JPanel();
     /** Step Panel */
     StepTypePanel stepTypePanel = new StepTypePanel(this);
     /** Component Panel */
     ComponentTypePanel componentTypePanel = new ComponentTypePanel(this);
     /** EHL Panel */
     EHLTypePanel EHLPanel = new EHLTypePanel(this, this.listModel);
     /** Dependency Panel */
     DependencyTypePanel dependencyPanel = new DependencyTypePanel(this);
     /** done button */
     JButton doneButton = new JButton();

    /**
     * a method to handle standard action events
     */
    public void actionPerformed(ActionEvent event) {
        Object object = event.getSource();
        if (object == stepTypePanel.stepEditButton)
            stepEditButton_actionPerformed(event); // edit button pressed
        else if (object == stepTypePanel.stepDeleteButton)
            stepDeleteButton_actionPerformed(event); // delete button pressed
        else if (object == stepTypePanel.stepClearButton)
            stepClearButton_actionPerformed(event); // clear button pressed
        else if (object == componentTypePanel.compEditButton) {
            if (ownerWindow == null)
                compEditButton_actionPerformed(event); // edit button pressed
        }
        else {

```

```

        int compIndex = ownerWindow.drawPanel.hitComponent;
        compEditButton_actionPerformed(event, compIndex); //edit button pressed
    }
}

        else if (object == componentTypePanel.compDeleteButton)
            compDeleteButton_actionPerformed(event); // delete button pressed
        else if (object == componentTypePanel.compClearButton)
            compClearButton_actionPerformed(event); // clear button pressed
        else if (object == EHLPanel.EHLAddButton)
            EHLAddButton_actionPerformed(event); // add button pressed
    else if (object == EHLPanel.updateButton)
        updateButton_actionPerformed(event); // update button pressed
        else if (object == EHLPanel.EHLEditButton)
            EHLEditButton_actionPerformed(event); // edit button pressed
        else if (object == EHLPanel.EHLDeleteButton)
            EHLDeleteButton_actionPerformed(event);
        else if (object == EHLPanel.EHLClearButton) // clear button pressed
            EHLClearButton_actionPerformed(event);
        else if (object == dependencyPanel.depenOKButton) // okay button pressed
            depenOKButton_actionPerformed(event);
        else if (object == dependencyPanel.depenCancelButton) // cancel button pressed
            depenCancelButton_actionPerformed(event);
        else if (object == dependencyPanel.depenUpdateButton) // update button pressed
            depenUpdateButton_actionPerformed(event);
        else if (object == doneButton) // done button pressed
            doneButton_actionPerformed(event);
        else if (object == EHLPanel.EHLDoneButton)
            EHLDoneButton_actionPerformed(event); // done button pressed
        else if (object == stepTypePanel.stepSaveButton)
            stepSaveButton_actionPerformed(event); // save button pressed
        if (object == componentTypePanel.compSaveButton)
            compSaveButton_actionPerformed(event); // save button pressed
        if (object == dependencyPanel.secondaryButton)
            secondaryButton_actionPerformed(event); // secondary button pressed
    }

    /**
     * this method handles the dependency update button event
     */
    public void depenUpdateButton_actionPerformed(ActionEvent event){
        for( int i=0; i< stepVector.size(); i++ ){
            StepType tempStepType =(StepType)stepVector.elementAt(i);
            String tempString = tempStepType.getStepID() ;
            String s = tempString.substring(2,3)+tempString.substring(4,tempString.length());
            dependencyPanel.depenStepComboBox.setSelectedIndex(i+1); // bypass the 0 element,because it is the header
            dependencyPanel.depenSecondaryTextField.setText(s);
        }
        depenOKButton_actionPerformed(null);
    }

    /**
     * Set the current project name for each panel's title
     *
     * @param selectedProject : the current project name
     */
    public void setProjectLabel( String selectedProject ){
        EHLPanel.projectLabel.setText( selectedProject );
        stepTypePanel.stepLabel.setText( selectedProject );
        componentTypePanel.compLabel.setText( selectedProject );
        dependencyPanel.depLabel.setText( selectedProject );
    }

    /**
     * Edit selected step type object and delete before add it into combo box and stepVector
     */
    public void stepEditButton_actionPerformed(ActionEvent event)
    {

```

```

        if( this.selectedStepIndex > 0 ){
            String stepName = stepTypePanel.stepNameTextField.getText();
            String stepDescription = stepTypePanel.stepDescriptionTextArea.getText();
            StepType tempStepType = (StepType)stepVector.get(this.selectedStepIndex-1);
            // controller portion of MVC
            tempStepType.setStepName(stepName);
            tempStepType.setStepDescription(stepDescription);

            this.setListModel( this.stepVector );
            this.setStepComboBox( this.stepVector );

            this.stepSaveButton_actionPerformed(null);
            saveTab[0] = false;
            setVisible( false );
            dispose();
        }

        //Clean up the frame
        this.stepClearButton_actionPerformed( null );
    }

/**
 * Edit selected step type object and delete before add it into combo box and stepVector
 */
    public String stepEditButton_actionPerformed(ActionEvent event, String stepID, int c1, int c2)
    {
        String stepName = stepTypePanel.stepNameTextField.getText();
        if( this.selectedStepIndex > 0 ){
            String stepDescription = stepTypePanel.stepDescriptionTextArea.getText();
            StepType tempStepType = (StepType)stepVector.get(this.selectedStepIndex-1);
            // controller portion of MVC
            tempStepType.setStepID(stepID);
            System.out.println(stepName);
            tempStepType.setStepName(stepName);
            tempStepType.setStepDescription(stepDescription);
            tempStepType.setStepFrom(c1);
            tempStepType.setStepTo(c2);

            this.setListModel( this.stepVector );
            this.setStepComboBox( this.stepVector );
            this.stepSaveButton_actionPerformed(null);
            saveTab[0] = false;
        }

        //Clean up the frame
        this.stepClearButton_actionPerformed( null );
        return stepName;
    }

/**
 * Edit selected step type object and delete before add it into combo box and stepVector
 */
    public String stepEditButton_actionPerformed(ActionEvent event, MyDefaultTableModel stepTable, int c1, int c2) {
        String versionString = "0";
        String stepName = stepTypePanel.stepNameTextField.getText();
        if( this.selectedStepIndex > 0 ){
            String stepID = stepTypePanel.stepIDTextField.getText();
            String stepDescription = stepTypePanel.stepDescriptionTextArea.getText();
            StepType tempStepType = (StepType)stepVector.get(this.selectedStepIndex-1);
            // controller portion of MVC
            tempStepType.setStepID(stepID);
            System.out.println(stepName);
            tempStepType.setStepName(stepName);
            tempStepType.setStepDescription(stepDescription);
            tempStepType.setStepFrom(c1);
            tempStepType.setStepTo(c2);
            ((StepQFD)tempStepType.getStepQFDHashtable().get(versionString)).setStepTable(stepTable);

```

```

        this.setListModel( this.stepVector );
        this.setStepComboBox( this.stepVector );
        this.stepSaveButton_actionPerformed(null);
        saveTab[0] = false;
    }

    //Clean up the frame
    this.stepClearButton_actionPerformed( null );
    return stepName;
}

/**
 * Delete selected step type object and remove it from combo box and stepVector
 */
public void stepDeleteButton_actionPerformed(ActionEvent event)
{
    if( this.selectedStepIndex > 0 ){
        this.stepVector.removeElementAt( this.selectedStepIndex - 1 );
        this.setListModel( this.stepVector );
        this.setStepComboBox( this.stepVector );
        saveTab[0] = false;
    }

    //Clean up the frame
    this.stepClearButton_actionPerformed( null );
    this.stepSaveButton_actionPerformed(null);
}

/**
 * Refresh all text fields in step panel
 */
public void stepClearButton_actionPerformed(ActionEvent event)
{
    {
        stepTypePanel.stepIDTextField.setText("");
        stepTypePanel.stepNameTextField.setText("");
        stepTypePanel.stepDescriptionTextArea.setText("");
        if( this.existedStepComboBox.getItemCount() > 0 ){
            this.existedStepComboBox.setSelectedIndex(0);
        }
    }
}

/**
 * Save all step type objects in stepVector in step.cfg file
 */
public void stepSaveButton_actionPerformed(ActionEvent event)
{
    try{
        FileOutputStream stepFileOut = new FileOutputStream( pathName+"\\step.cfg" );
        ObjectOutputStream stepOut= new ObjectOutputStream( stepFileOut );

        if( this.stepVector.size()> 0 ){
            if(stepOut != null ){
                stepOut.writeObject( this.stepVector );
                this.setStepComboBox( this.stepVector );
                saveTab[0] = true;
            }

            stepOut.flush();
            stepOut.close();
            stepFileOut.close();
        }
    }
    catch( FileNotFoundException fe ){
        debug("FileNotFoundException: "+fe);
    }
    catch( IOException e ){
        debug("IOException: "+e);
    }
}

//Clean up the frame

```



```

        this.stepClearButton_actionPerformed( null );
    }

/**
 * Edit selected component type object and delete before add it into combo box and compVector
 */
    public void compEditButton_actionPerformed(ActionEvent event)
    {
        if( this.selectedCompIndex > 0 ){
            String compID = componentTypePanel.compIDTextField.getText();
            String compName = componentTypePanel.compNameTextField.getText();
            String compDescription = componentTypePanel.compDescriptionTextArea.getText();
            ComponentType tempType = (ComponentType)compVector.get(this.selectedCompIndex-1);
            // controller portion of MVC
            tempType.setComponentName(compName);
            tempType.setComponentDescription(compDescription);

            this.setCompComboBox( this.compVector );
            this.compSaveButton_actionPerformed(null);
            saveTab[1] = false;
            setVisible( false );
            dispose();
        }

        //Clean up the frame
        this.compClearButton_actionPerformed( null );
    }

/**
 * Edit selected component type object and delete before add it into combo box and compVector
 */
    public void compEditButton_actionPerformed(ActionEvent event, int compIndex)
    {
        String compName = componentTypePanel.compNameTextField.getText();
        if( this.selectedCompIndex > 0 ){
            String compID = componentTypePanel.compIDTextField.getText();
            String compDescription = componentTypePanel.compDescriptionTextArea.getText();
            ComponentType tempType = (ComponentType)compVector.get(this.selectedCompIndex-1);
            // controller portion of MVC
            tempType.setComponentName(compName);
            tempType.setComponentDescription(compDescription);
            tempType.setComponentValue(compIndex);

            this.setCompComboBox( this.compVector );
            this.compSaveButton_actionPerformed(null);
            saveTab[1] = false;
        }

        //Clean up the frame
        this.compClearButton_actionPerformed( null );
    }
    // return compName;
    setVisible( false );
    dispose();
}

/**
 * Edit selected component type object and delete before add it into combo box and compVector
 * Used to edit components for house of QFD
 */
    public void compEditButton_actionPerformed(ActionEvent event, MyDefaultTableModel compTable, int compIndex, int
depIndex) {
        String versionString = "0";

        if( this.selectedCompIndex > 0 ){
            String compID = componentTypePanel.compIDTextField.getText();
            String compName = componentTypePanel.compNameTextField.getText();
            String compDescription = componentTypePanel.compDescriptionTextArea.getText();
            ComponentType tempType = (ComponentType)compVector.get(this.selectedCompIndex-1);

```

```

// controller portion of MVC
tempType.setComponentName(compName);
tempType.setComponentDescription(compDescription);
tempType.setComponentValue(compIndex);
((ComponentQFD)tempType.getComponentHashtable().get(versionString)).setComponentTable(compTable,depIndex);

        this.setCompComboBox( this.compVector );
this.compSaveButton_actionPerformed(null);
        saveTab[1] = false;
    }

        //Clean up the frame
        this.compClearButton_actionPerformed( null );
setVisible( false );
dispose();
    }

/**
 * Delete selected component type object and remove it from combo box and compVector
 */
    public void compDeleteButton_actionPerformed(ActionEvent event)
    {
        if( this.selectedCompIndex > 0 ){
            this.compVector.removeElementAt( this.selectedCompIndex - 1 );
            this.setCompComboBox( this.compVector );
            saveTab[1] = false;
        }

        //Clean up the frame
        this.compClearButton_actionPerformed( null );
this.compSaveButton_actionPerformed(null);
    }

/**
 * Refresh all text fields in component panel
 */
    public void compClearButton_actionPerformed(ActionEvent event)
    {
        componentTypePanel.compIDTextField.setText("");
        componentTypePanel.compNameTextField.setText("");
        componentTypePanel.compDescriptionTextArea.setText("");
        if( this.existedCompComboBox.getItemCount() > 0 ){
            this.existedCompComboBox.setSelectedIndex(0);
        }
    }

/**
 * Save all component type objects in compVector in component.cfg file
 */
    public void compSaveButton_actionPerformed(ActionEvent event)
    {
        ObjectFileOperations compSaveOFO = new ObjectFileOperations();
        compSaveOFO.fileSaveActionPerformed(pathName+"\\\"+projectName+"\\component.cfg", this.compVector);
        if( this.compVector.size()> 0 ){
            saveTab[1] = true;
            this.setCompComboBox( this.compVector );
        }

        //Clear the Frame
        this.compClearButton_actionPerformed( null );
    }

/**
 * Add new EHL object into combo box and EHLVector
 */
    public void EHLAddButton_actionPerformed(ActionEvent event)
    {
        String EHLName = EHLPanel.EHLNameTextField.getText();

```

```

String EHLPath = EHLPanel.EHLPathTextField.getText();

StringTokenizer st = new StringTokenizer( EHLPath, " , " );
Vector tokenizeVector = new Vector();
while( st.hasMoreTokens() ){
    tokenizeVector.addElement( st.nextToken() );
}

if( tokenizeVector.size()< 2 ){
    JOptionPane.showMessageDialog(this, "The loop must have at least 2 steps!",
        "Error", JOptionPane.ERROR_MESSAGE);
    return;
}
else{
    EHL ehl = new EHL( EHLName, EHLPath );
    this.EHLVector.addElement( ehl );
    this.setEHLComboBox( this.EHLVector );
    saveTab[2] = false;

    //Clean up the frame
    this.EHLClearButton_actionPerformed( null );
}
}

/**
 * Edit selected EHL object and delete before add it into combo box and EHLVector
 */
public void EHLEditButton_actionPerformed(ActionEvent event)
{
    if( this.selectedEHLIndex > 0 ){
        String EHLName = EHLPanel.EHLNameTextField.getText();
        String EHLPath = EHLPanel.EHLPathTextField.getText();
        EHL ehl = new EHL( EHLName, EHLPath );
        this.EHLVector.setElementAt( ehl, this.selectedEHLIndex-1 );
        this.setEHLComboBox( this.EHLVector );
        saveTab[2] = false;
    }

    //Clean up the frame
    this.EHLClearButton_actionPerformed( null );
}

/**
 * Delete selected EHL object and remove it from combo box and EHLVector
 */
public void EHLDeleteButton_actionPerformed(ActionEvent event)
{
    if( this.selectedEHLIndex > 0 ){
        this.EHLVector.removeElementAt( this.selectedEHLIndex - 1 );
        this.setEHLComboBox( this.EHLVector );
        saveTab[2] = false;
    }

    //Clean up the frame
    this.EHLClearButton_actionPerformed( null );
}

/**
 * Refresh all text fields in EHL panel
 */
public void EHLClearButton_actionPerformed(ActionEvent event)
{
    EHLPanel.EHLNameTextField.setText("");
    EHLPanel.EHLPathTextField.setText("");
}

/**
 * Save all EHL objects in EHLVector in loop.cfg file and go directly into dependency panel

```

```

*/
    public void EHLDoneButton_actionPerformed(ActionEvent event)
    {
try{
    FileOutputStream EHLFileOut = new FileOutputStream( pathName+"\\ "+projectName+"\\loop.cfg" );
    ObjectOutputStream EHLOut = new ObjectOutputStream( EHLFileOut );

        if( this.EHLVector.size(> 0 ){
            if(EHLOut != null ){
                EHLOut.writeObject( this.EHLVector );
                saveTab[2] = true;
                this.readDepFile();
                this.setDepenEHLComboBox( this.EHLVector);

                //Only use for EHL tabbedPane
                this.setEnabled( 3, true);
                this.configManagTabbedPane.setSelectedIndex( 3 );

                this.setEnabled( 0, false );
                this.setEnabled( 1, false );
                this.setEnabled( 2, true );
            }
            EHLOut.flush();
            EHLOut.close();
            EHLFileOut.close();
        }
    }
    catch( FileNotFoundException fe ){
        debug("FileNotFoundException: "+fe);
    }
    catch( IOException e ){
        debug("IOException: "+e);
    }
    }

    //Clean up the frame
    this.EHLClearButton_actionPerformed( null );
    }

/**
 * Save all dependency objects in depenHashtable in dependency.cfg file
 * Before saving it, check all secondary inputs of each dependency to be set or not.
 * The warning message will show up if one of dependency objects
 * didn't set the secondary input
 */
    public void depenOKButton_actionPerformed(ActionEvent event)
    {
saveLastDep();

        Vector v = new Vector();
        Vector v2 = new Vector();
        Dependency dep = null;

        Enumeration enum = (Enumeration)listModel.elements();
        while( enum.hasMoreElements() ){
            if( !this.depenHashtable.containsKey((String)enum.nextElement()) ){
                v.addElement("");
                v2.addElement("");
            }
        }
        enum = null;
        try{
            FileOutputStream depFileOut = new FileOutputStream( pathName+"\\ "+projectName+"\\dependency.cfg" );
            ObjectOutputStream dependencyOut = new ObjectOutputStream( depFileOut );
            if( this.depenHashtable.size() > 0 ){
                enum = this.depenHashtable.elements();
                while( enum.hasMoreElements() ){
                    Dependency d = (Dependency)enum.nextElement();
                    if( d!= null ){

```

```

        v.addElement(((String)d.getSecondaryInput()).trim());
        v2.addElement(((String)d.getStep()).trim());
    }
    else{
        v.addElement("");
        v2.addElement("");
    }
}

for( int i=0; i<v.size(); i++ ){
    if( ((String)v.elementAt(i)).equals("") ){
        if( !((String)v2.elementAt(i)).equals(STEP_TYPE_TITLE) ){
            JOptionPane.showMessageDialog(this, "Can not process since \n some steps do not have the secondary \n input
component types!",
                                "Error", JOptionPane.ERROR_MESSAGE);
        }
        return;
    }
}

if( dependencyOut != null ){
    dependencyOut.writeObject( this.depenHashtable );
    saveTab[3] = true;
}
dependencyOut.flush();
dependencyOut.close();
depFileOut.close();
}

}
catch( FileNotFoundException fe ){
    debug("FileNotFoundException_DepFileOut: "+fe);
}
catch( IOException e ){
    debug("IOException: "+e);
}

this.setEnabled( 3, false);
this.configManagTabbedPane.setSelectedIndex( 0 );
this.setEnabled( 0, true );
this.setEnabled( 1, true );
this.setEnabled( 2, false );
dependencyPanel.depenPrimaryTextField.setText("");
dependencyPanel.depenOutputTextField.setText("");
dependencyPanel.depenSecondaryTextField.setText("");
}

/**
 * Before leave the dependency panel, save all the dependency objects in depenHashtable
 */
public void depenCancelButton_actionPerformed(ActionEvent event)
{
    ObjectFileOperations depOFO = new ObjectFileOperations();
    depOFO.fileSaveActionPerformed( pathName+"\\ "+projectName+"\\dependency.cfg", this.depenHashtable);
}
/*
try{
    FileOutputStream depFileOut = new FileOutputStream( pathName+"\\ "+projectName+"\\dependency.cfg" );
    ObjectOutputStream dependencyOut = new ObjectOutputStream( depFileOut );
    if( this.depenHashtable.size() > 0 ){
        if( dependencyOut != null ){
            dependencyOut.writeObject( this.depenHashtable );
        }
        dependencyOut.flush();
        dependencyOut.close();
        depFileOut.close();
    }
}
catch(IOException e ){
    debug("IOException_DepCancel: "+e);
}
*/

```

```

        this.setEnabled( 3, false);

        this.configManagTabbedPane.setSelectedIndex( 0 );

        this.setEnabled( 0, true );
        this.setEnabled( 1, true );
        this.setEnabled( 2, false );

        dependencyPanel.depenPrimaryTextField.setText("");
        dependencyPanel.depenOutputTextField.setText("");
        dependencyPanel.depenSecondaryTextField.setText("");
    }

/**
 * Check all the changes had saved or not before exit ProjectSchemaFrame
 */
    public void doneButton_actionPerformed(ActionEvent event)
    {
        for( int i=0; i<saveTab.length; i++ ){
            if( !saveTab[i] ){
                Object[] options = { "Yes", "No" };
                int j = JOptionPane.showOptionDialog(this, "Would you like to save?",
                "Warning", JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE, null, options, options[0]);
                if( j==0 ){
                    stepSaveButton_actionPerformed(event);
                    compSaveButton_actionPerformed(event);
                    EHL.DoneButton_actionPerformed(event);
                    depenOKButton_actionPerformed(event);

                }
            }
        }

        setVisible( false );
        dispose();
    }

/**
 * Set enable a tab panel
 *
 * @param tabIndex : an index of panel
 * @param flag : true or false
 */
    public void setEnabled( int tabIndex, boolean flag ){
        this.configManagTabbedPane.setEnabledAt( tabIndex, flag );
    }

/**
 * Short cut to print the output
 * @param string : the output string
 */
    public void debug(String string ){
        System.out.println(string);
    }

/**
 * a method to handle item state changes for comboboxes
 */
    public void itemStateChanged( ItemEvent event)    {
        Object object = event.getSource();
        if (object == dependencyPanel.depenStepComboBox)
            depenStepComboBox_itemStateChanged(event);
        else if (object == dependencyPanel.depenEHLComboBox)
            depenEHLComboBox_itemStateChanged(event);
        else if (object == this.existedStepComboBox)
            existedStepComboBox_itemStateChanged(event);
        else if (object == this.existedCompComboBox)
            existedCompComboBox_itemStateChanged(event);
        else if (object == EHLPanel.existedEHLComboBox)

```

```

        existedEHLComboBox_itemStateChanged(event);
    }

/**
 * Refresh stepHashtable and existedStepComboBox, and add new items
 *
 * @param stepVector : contains the current step type objects
 */
public void setStepComboBox(Vector stepVector ){
    this.existedStepComboBox.removeAllItems();
    this.existedStepComboBox.addItem(STEP_TYPE_TITLE);

    this.stepHashtable = new Hashtable();

    for( int i=0; i< stepVector.size(); i++ ){
        StepType st = (StepType)stepVector.elementAt( i );
        String stID = st.getStepID();

        //set step Hashtable
        this.stepHashtable.put( stID, st );

        this.existedStepComboBox.addItem(stID);
    }
}

/**
 * Refresh stepHashtable and existedStepComboBox, and add new items
 *
 * @param stepVector : contains the current step type objects
 */
public void setStepComboBox(Vector stepVector, GraphPanel drawPanel ){
    this.existedStepComboBox.removeAllItems();
    this.existedStepComboBox.addItem(STEP_TYPE_TITLE);

    this.stepHashtable = new Hashtable();

    for( int i=0; i< stepVector.size(); i++ ){
        StepType st = (StepType)stepVector.elementAt( i );
        String stID = st.getStepID();
        // set steps in drawPanel

        drawPanel.stepUpdate(st.getStepFrom(),st.getStepTo(),50);
        drawPanel.stepName[st.getStepFrom()][st.getStepTo()]=st.getStepName();
        drawPanel.stepID[st.getStepFrom()][st.getStepTo()]=st.getStepID();
        drawPanel.repaint();

        //set step Hashtable
        this.stepHashtable.put( stID, st );

        this.existedStepComboBox.addItem(stID);
    }
}

/**
 * Refresh compHashtable and existedCompComboBox, and add new items
 *
 * @param compVector : contains the current component type objects
 */
public void setCompComboBox(Vector compVector ){
    this.existedCompComboBox.removeAllItems();
    this.existedCompComboBox.addItem(COMPONENT_TYPE_TITLE);

    this.compHashtable = new Hashtable();

    for( int i=0; i< compVector.size(); i++ ){
        ComponentType ct = (ComponentType)compVector.elementAt( i );

        String ctID = ct.getComponentID();

```

```

        //set step Hashtable
        this.compHashtable.put( ctID, ct );

        this.existedCompComboBox.addItem(ctID);
    }
}

/**
 * Refresh compHashtable and existedCompComboBox, and add new items
 *
 * @param compVector : contains the current component type objects
 */
public void setCompComboBox(Vector compVector, GraphPanel drawPanel ){
    this.existedCompComboBox.removeAllItems();
    this.existedCompComboBox.addItem(COMONENT_TYPE_TITLE);

    this.compHashtable = new Hashtable();

    for( int i=0; i< compVector.size(); i++){
        ComponentType ct = (ComponentType)compVector.elementAt( i );

        //set information in drawPanel
        drawPanel.component[ct.getComponentValue()]=ct.getComponentPosition();
        drawPanel.componentID[ct.getComponentValue()]=ct.getComponentID();
        drawPanel.repaint();

        String ctID = ct.getComponentID();

        //set step Hashtable
        this.compHashtable.put( ctID, ct );

        this.existedCompComboBox.addItem(ctID);
    }
}

/**
 * Refresh EHLHashtable and existedEHLComboBox, and add new items
 *
 * @param EHLVector : contains the current EHL objects
 */
public void setEHLComboBox(Vector EHLVector ){
    EHLPanel.existedEHLComboBox.removeAllItems();
    EHLPanel.existedEHLComboBox.addItem(PROCESS_TITLE);

    this.EHLHashtable = new Hashtable();

    for( int i=0; i< EHLVector.size(); i++){
        EHL ehl = (EHL)EHLVector.elementAt( i );
        String EHLName = ehl.getEHLName();

        //set step Hashtable
        this.EHLHashtable.put( EHLName, ehl );

        EHLPanel.existedEHLComboBox.addItem(EHLName);
    }
}

/**
 * Refresh depenEHLComboBox and add new items
 *
 * @param EHLVector : contains the current EHL objects
 */
public void setDepenEHLComboBox(Vector EHLVector ){
    dependencyPanel.depenEHLComboBox.removeAllItems();

    for( int j=0; j< EHLVector.size(); j++){
        EHL ehl = (EHL)EHLVector.elementAt(j);
        dependencyPanel.depenEHLComboBox.addItem(ehl.getEHLName());
    }
}

```



```

    }
}

/**
 * Refresh depenStepComboBox, add new items,
 * and create new directory for the selected step
 *
 * @param stepVector : contains the current step names
 */
public void setDependStepComboBox(Vector stepVector ){
    String loopName = (String)dependencyPanel.depenEHLComboBox.getSelectedItem();
    dependencyPanel.depenStepComboBox.removeAllItems();
    dependencyPanel.depenStepComboBox.addItem(STEP_TYPE_TITLE);
    for( int i=0; i< stepVector.size(); i++){
        String depStep = ((String)stepVector.elementAt( i )).trim();
        dependencyPanel.depenStepComboBox.addItem(depStep);
        File directory = new File( this.pathName+"\\\\"+projectName, depStep );
        if( !directory.isDirectory() ){
            directory.mkdir();
        }
    }
}

/**
 * View the selected step type object
 *
 * @param st : selected step type object
 */
public void setStepInfo(StepType st ){
    stepTypePanel.stepIDTextField.setText( st.getStepID() );
    stepTypePanel.stepNameTextField.setText( st.getStepName() );
    stepTypePanel.stepDescriptionTextArea.setText( st.getStepDescription() );
}

/**
 * View the selected component type object
 *
 * @param ct : selected component type object
 */
public void setCompInfo(ComponentType ct ){
    componentTypePanel.compIDTextField.setText( ct.getComponentID() );
    componentTypePanel.compNameTextField.setText( ct.getComponentName() );
    componentTypePanel.compDescriptionTextArea.setText( ct.getComponentDescription() );
}

/**
 * View the selected EHL object
 *
 * @param ehl : selected EHL object
 */
public void setEHLInfo(EHL ehl ){
    EHLPanel.EHLNameTextField.setText( ehl.getEHLName() );
    EHLPanel.EHLPathTextField.setText( ehl.getEHLPath() );
}

/**
 * sets the component name and ID based on the input from the
 * schema diagram
 */
public void setCompNameTextField (String compName) {
    componentTypePanel.compNameTextField.setText(compName);
    componentTypePanel.compIDTextField.setText(compName);
    configManagTabbedPane.setSelectedIndex(1);
}

/**
 * sets the step name and ID based on the user input from the
 * schema diagram

```

```

/**
public void setStepNameTextField (String stepName, String compName) {
    stepTypePanel.stepNameTextField.setText(stepName);
    stepTypePanel.stepIDTextField.setText("s-"+compName);
    configManagTabbedPane.setSelectedIndex(0);
}

/**
 * View the selected dependency object
 *
 * @param selectedDepStep : selected dependency object name
 */
public void setDepInfo(String selectedDepStep ){
    if( !selectedDepStep.equals(STEP_TYPE_TITLE) ){
        StringTokenizer st = new StringTokenizer( selectedDepStep,"-");
        String before_ = st.nextToken("-");
        String _after = st.nextToken("-");
        _after = _after.substring(1,2) + "Component";
        dependencyPanel.depenPrimaryTextField.setText( _after );
        dependencyPanel.depenOutputTextField.setText( _after);

        if( this.depenHashtable.containsKey( selectedDepStep ) ){
            Dependency dep = (Dependency)this.depenHashtable.get( selectedDepStep );
            String secondInput = dep.getSecondaryInput();
            if( !secondInput.equals("") ){
                dependencyPanel.depenSecondaryTextField.setText(secondInput);
            }
        }
    }
}

/**
 * Read step.cfg, component.cfg, and loop.cfg files and insert their contents
 * into stepVector, compVector, and EHLVector respectively
 *
 * @param projectName : the name of current project
 */
public void readInputFiles(String projectName ){
    ObjectFileOperations tempOFO = new ObjectFileOperations ();
    this.compVector = tempOFO.fileLoadActionPerformed(pathName+"\\ "+projectName, "component.cfg");
    if( this.compVector.size() > 0 ){
        this.existedCompComboBox.removeAllItems();
        this.existedCompComboBox.addItem(COMONENT_TYPE_TITLE);
        this.compHashtable = new Hashtable();

        for( int i=0; i< compVector.size(); i++ ){
            ComponentType ct = (ComponentType)compVector.elementAt( i );
            String ctID = ct.getComponentID();
            //set step Hashtable
            this.compHashtable.put( ctID, ct );
            this.existedCompComboBox.addItem(ctID);
        }
    }

    this.stepVector=tempOFO.fileLoadActionPerformed(pathName+"\\ "+projectName, "step.cfg");
    if( this.stepVector.size() > 0 ){
        this.existedStepComboBox.removeAllItems();
        this.existedStepComboBox.addItem(STEP_TYPE_TITLE);
        this.stepHashtable = new Hashtable();
        for( int i=0; i< stepVector.size(); i++ ){
            StepType st = (StepType)stepVector.elementAt( i );
            String stID = st.getStepID();
            //set step Hashtable
            this.stepHashtable.put( stID, st );
            this.existedStepComboBox.addItem(stID);
        }
    }
    this.setListModel(this.stepVector);
}

```

```

        this.EHLVector=tempOFO.fileLoadActionPerformed(pathName+"\\ "+projectName, "loop.cfg");
        if( this.EHLVector.size() > 0 ){
            this.setEHLComboBox( this.EHLVector );
        }
    }

    /**
     * Read step.cfg, component.cfg, and loop.cfg files and insert their contents
     * into stepVector, compVector, and EHLVector respectively
     *
     * @param pathName : the path of current project
     */
    public void readInputFiles(GraphPanel drawPanel){
        ObjectFileOperations ofo = new ObjectFileOperations();
        Vector tempVector = new Vector();
        tempVector = ofo.fileLoadActionPerformed(pathName+"\\ "+projectName,"depAttrib.cfg");
        if( tempVector.size() >0 ) { // store dependencies in draw panel dep vector
            drawPanel.depAttrib=tempVector;
            ownerWindow.drawToolBar.setCalcButton(true);
        }
        this.compVector=ofo.fileLoadActionPerformed(pathName+"\\ "+projectName, "component.cfg");
        if( this.compVector.size() > 0 ){
            this.existedCompComboBox.removeAllItems();
            this.existedCompComboBox.addItem(COMONENT_TYPE_TITLE);
            this.compHashtable = new Hashtable();
            for( int i=0; i< compVector.size(); i++){
                ComponentType ct = (ComponentType)compVector.elementAt( i );

                // MVC
                ct.addObserver(drawPanel);

                //set information in drawPanel
                drawPanel.component[ct.getComponentValue()]=ct.getComponentPosition();
                String ctID = ct.getComponentID();
                //set step Hashtable
                this.compHashtable.put( ctID, ct );
                this.existedCompComboBox.addItem(ctID);
            }
        }
        this.stepVector=ofo.fileLoadActionPerformed(pathName+"\\ "+projectName, "step.cfg");
        if( this.stepVector.size() > 0 ){
            this.existedStepComboBox.removeAllItems();
            this.existedStepComboBox.addItem(STEP_TYPE_TITLE);
            this.stepHashtable = new Hashtable();
            for( int i=0; i< stepVector.size(); i++){
                StepType st = (StepType)stepVector.elementAt( i );

                // MVC
                st.addObserver(drawPanel);

                String stID = st.getStepID();
                // set steps in drawPanel
                //set step Hashtable
                this.stepHashtable.put( stID, st );
                drawPanel.stepUpdate(st.getStepFrom(),st.getStepTo(),50);

                this.existedStepComboBox.addItem(stID);
            }
        }
        this.setListModel(this.stepVector);
    }
    this.EHLVector=ofo.fileLoadActionPerformed(pathName+"\\ "+projectName, "loop.cfg");
    if( this.EHLVector.size() > 0 ){
        this.setEHLComboBox( this.EHLVector );
    }
}

    /**
     * Read dependency.cfg file and insert its content into depenHashtable
     */
    public void readDepFile(){

```

```

File dependencyFile = new File(pathName+"\\ "+projectName, "dependency.cfg");
if( dependencyFile.exists() ){
    try{
        FileInputStream fileInput = new FileInputStream( dependencyFile );
        ObjectInputStream dependencyIn = new ObjectInputStream( fileInput );
        if( dependencyIn != null ){
            this.depenHashtable = (Hashtable)dependencyIn.readObject();
        }
        dependencyIn.close();
        fileInput.close();
    }
    catch( IOException e ){
        debug("IOException_readDepIn: "+e);
    }
    catch( ClassNotFoundException ex ){
        debug("ClassNotFoundException_readDepIn: "+ex);
    }
}
}

/**
 * List all existing step type objects in the project
 * and allow a user to view or edit them
 */
void existedStepComboBox_itemStateChanged(ItemEvent event)
{
    if( event.getStateChange() == ItemEvent.SELECTED ){
        String selectedStepID = (String)event.getItem();
        this.selectedStepIndex = this.existedStepComboBox.getSelectedIndex();

        if( this.selectedStepIndex == 0 ){
            //Clean up the frame
            this.stepClearButton_actionPerformed( null );
        }
        else if( this.selectedStepIndex > 0 ){
            if( this.stepHashtable.containsKey( selectedStepID ) ){
                this.setStepInfo( (StepType)this.stepHashtable.get( selectedStepID ) );
            }
        }
    }
}

/**
 * List all existing component type objects in the project
 * and allow a user to view or edit them
 */
void existedCompComboBox_itemStateChanged(ItemEvent event)
{
    if( event.getStateChange() == ItemEvent.SELECTED ){
        String selectedCompID = (String)event.getItem();
        this.selectedCompIndex = this.existedCompComboBox.getSelectedIndex();
        if( this.selectedCompIndex == 0 ){
            //Clean up the frame
            this.compClearButton_actionPerformed( null );
        }
        if( this.compHashtable.containsKey( selectedCompID ) ){
            this.setCompInfo( (ComponentType)this.compHashtable.get( selectedCompID ) );
        }
    }
}

/**
 * List all existing EHL objects in the project
 * and allow a user to view or edit them
 */
void existedEHLComboBox_itemStateChanged(ItemEvent event)
{
    if( event.getStateChange() == ItemEvent.SELECTED ){

```

```

String selectedEHLName = (String)event.getItem();
this.selectedEHLIndex = EHLPanel.existedEHLComboBox.getSelectedIndex();
if( this.selectedEHLIndex == 0 ){
    //Clean up the frame
    this.EHLClearButton_actionPerformed( null );
}
if( this.EHLHashtable.containsKey( selectedEHLName ) ){
    this.setEHLInfo( (EHL)this.EHLHashtable.get( selectedEHLName ) );
}
}
}

/**
 * List all existing step type objects in a selected project
 * and allow a user to view, edit, or set secondary input
 */
void depenStepComboBox_itemStateChanged(ItemEvent event)
{
    Dependency dep = null;
    String selectedDepStep = null;
    if( event != null ){
        selectedDepStep = ((String)event.getItem()).trim();
    }
    this.testIndex++;
    // dependency information related to primary and secondary input components
    String loopName = (String)dependencyPanel.depenEHLComboBox.getSelectedItem();
    String stepName = (String)dependencyPanel.depenStepComboBox.getItemAt(this.selectedDepenStepIndex);
    String output = (String)dependencyPanel.depenOutputTextField.getText();
    /** string for primary input component */
    String primary = (String)dependencyPanel.depenPrimaryTextField.getText();
    /** string for secondary input component */
    String secondary = (String)dependencyPanel.depenSecondaryTextField.getText();

    if( this.testIndex < 2 ){
        this.selectedDepenStepIndex = dependencyPanel.depenStepComboBox.getSelectedIndex();
        if( (secondary != null) && (this.selectedDepenStepIndex > 0) ){
            if( this.depenHashtable.containsKey( stepName ) ){
                dep = (Dependency) this.depenHashtable.get( stepName );
                if( !secondary.equals("") ){
                    dep.setSecondaryInput( secondary );
                }
            }
            else{
                dep = new Dependency( loopName, stepName, output, primary, secondary);
                this.depenHashtable.put( stepName, dep );
            }
            dependencyPanel.depenSecondaryTextField.setText("");
        }
    }
    else {
        if( (stepName != null) && (this.depenHashtable.containsKey(stepName)) ){
            dep = (Dependency) this.depenHashtable.get( stepName );
            if( !secondary.equals("") ){
                dep.setSecondaryInput( secondary );
            }
        }
        this.testIndex = 0;
    }
    if( selectedDepStep.equals(STEP_TYPE_TITLE) ){
        dependencyPanel.depenOutputTextField.setText("");
        dependencyPanel.depenPrimaryTextField.setText("");
        dependencyPanel.depenSecondaryTextField.setText("");
    }
    else{
        this.setDepInfo( selectedDepStep );
    }
}
}

```

```

/**
 * List all existing EHL name in the project
 * and allow a user to see its all step type objects
 */
void depenEHLComboBox_itemStateChanged(ItemEvent event)
{
    if( event.getStateChange() == ItemEvent.SELECTED ){
        String selectedItem = (String)event.getItem();
        int selectedIndex = dependencyPanel.depenEHLComboBox.getSelectedIndex();

        if( this.EHLHashtable.containsKey( selectedItem ) ){
            EHL ehl = (EHL)this.EHLHashtable.get( selectedItem );
            StringTokenizer st = new StringTokenizer( (String)ehl.getEHLPath(), "," );
            Vector tokenizeVector = new Vector();
            while( st.hasMoreTokens() ){
                tokenizeVector.addElement( st.nextToken() );
            }
            this.setDependStepComboBox( tokenizeVector );
        }
    }
}

/**
 * Set step type object list in EHL panel
 *
 * @param stepVector : all existing step type objects
 */
public void setListModel(Vector stepVector ){
    this.listModel.removeAllElements();
    for( int i=0; i< stepVector.size(); i++ ){
        StepType st = (StepType) stepVector.elementAt( i );
        this.listModel.addElement( st.getStepID() );
    }
}

// standard mouse events for future functionality
public void mousePressed( MouseEvent event)    {}
public void mouseReleased( MouseEvent event)   {}
public void mouseEntered( MouseEvent event)    {}
public void mouseExited( MouseEvent event)     {}

/**
 * method to handle mouse clicked event
 */
public void mouseClicked( MouseEvent event)    {
    Object object = event.getSource();
    if (object == EHLPanel.stepTypesList)
        stepTypesList_mouseClicked(event);
}

/**
 * Monitor mouse click action on stepTypesList
 */
void stepTypesList_mouseClicked(MouseEvent event)
{
    String s = ((String)EHLPanel.stepTypesList.getSelectedValue()).trim();
    String EHLStep = (String) EHLPanel.EHLPathTextField.getText();
    Vector v = new Vector();
    if( !EHLStep.equals("") ){
        StringTokenizer st = new StringTokenizer(EHLStep, ",");
        while( st.hasMoreTokens() ){
            String s1 = ((String) st.nextToken()).trim();
            v.addElement(s1);
        }
    }
    if( !v.contains(s) ){
        EHLPanel.EHLPathTextField.setText(EHLStep+","+s);
        v.addElement(s);
    }
}

```

```

        else{
            JOptionPane.showMessageDialog(this, s+" is already in this process!",
                "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
    else{
        EHLPanel.EHLPathTextField.setText(s);
    }
}

/**
 * updates all fields automatically
 */
public void updateButton_actionPerformed(ActionEvent event)    {
    Vector v = new Vector();
    for (int i=0; i<stepVector.size(); i++) {
        String EHLStep = (String) EHLPanel.EHLPathTextField.getText();
        StepType tempST = (StepType)stepVector.elementAt(i);
        String s = tempST.getStepID();
        if( !EHLStep.equals("") ){
            StringTokenizer st = new StringTokenizer(EHLStep, ",");
            while( st.hasMoreTokens() ){
                String s1 = ((String) st.nextToken()).trim();
                v.addElement(s1);
            }
            if( !v.contains(s) ){
                EHLPanel.EHLPathTextField.setText(EHLStep+", "+s);
                v.addElement(s);
            }
        }
        else{
            EHLPanel.EHLPathTextField.setText(s);
        }
    }
    EHLAddButton_actionPerformed(null);
    EHLDoneButton_actionPerformed(null);
}

/**
 * Save the last selected dependency object with
 * its content before quit the dependency panel
 */
void saveLastDep(){
    Dependency dep = null;
    // dependency data related to primary and secondary input/output components
    String loopName = (String)dependencyPanel.depenEHLComboBox.getSelectedItem();
    String stepName = (String)dependencyPanel.depenStepComboBox.getItemAt(this.selectedDepenStepIndex);
    String output = (String)dependencyPanel.depenOutputTextField.getText();
    String primary = (String)dependencyPanel.depenPrimaryTextField.getText();
    String secondary = (String)dependencyPanel.depenSecondaryTextField.getText();

    this.selectedDepenStepIndex = dependencyPanel.depenStepComboBox.getSelectedIndex();
    if( (secondary != null) && (this.selectedDepenStepIndex > 0) ){
        if( this.depenHashtable.containsKey( stepName ) ){
            dep = (Dependency) this.depenHashtable.get( stepName );
            if( !secondary.equals("") ){
                dep.setSecondaryInput( secondary );
            }
        }
        else{
            dep = new Dependency( loopName, stepName, output, primary, secondary);
            this.depenHashtable.put( stepName, dep );
        }
    }
}

/**
 * Launch ListDialog where lists all existing component type names

```

```

    */
    void secondaryButton_actionPerformed(ActionEvent event) {
        (new ListDialog(this, "Component Types List", this.compVector)).setVisible(true);
    }

    /**
     * Set all selected component type names from ListDialog
     * into secondary input text field in dependency panel
     *
     * @param objs : current selected items from ListDialog
     */
    public void setItemList(Object[] objs){
        if( objs.length >= 1 ){
            String s = (String)objs[0];
            for( int i=1; i<objs.length; i++){
                s = s + ", " + objs[i];
            }
            dependencyPanel.depenSecondaryTextField.setText(s);
        }
    }
}

```

18. Cases.ReviewComponentContentDialog

```

    /**-----
     * @Filename: ReviewComponentContentDialog.java
     * @Date: 3-7-2003
     * @Author: Le Hahn but modified by Arthur Clomera for NPS Thesis
     * Description: rView all links in a component content of selected step
     *
     * Implement CasesTitle where stores all global variables of Cases package
     * Modified with standard event handlers and listeners
     * Removed dependencies upon Visual Cafe
     * @Compiler: JDK 1.3.1
     * -----
    */
package Cases;

import java.awt.*;

import java.awt.event.ActionListener;

import java.awt.event.ItemListener;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.util.Vector;

import javax.swing.*;

////////////////////////////////////
    /**
     * ReviewComponentContentDialog : View all links in a component content of selected step
     *
     * Implement CasesTitle where stores all global variables of Cases package
     */
    //////////////////////////////////////
    public class ReviewComponentContentDialog extends javax.swing.JDialog implements CasesTitle, ActionListener, ItemListener,
    MouseListener
    {
        /**
         * listModel : list all links of a selected link type
         */
    }

```



```

DefaultListModel listModel = new DefaultListModel();

/**
 * storedVector : contains links of all available link types in the
 * component content of selected step
 */
public Vector[] storedVector = {new Vector(), new Vector(), new Vector(), new Vector(), new Vector(), new Vector()};

/**
 * Build ReviewComponentContentDialog
 */
public ReviewComponentContentDialog(Frame parent) {
    super(parent);

    //{{INIT_CONTROLS
    setTitle("Review Component Content");
    setModal(true);
    getContentPane().setLayout(null);
    setSize(500,450);
    setVisible(false);
    JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
    JLabel1.setText("Available Links");
    getContentPane().add(JLabel1);
    JLabel1.setForeground(java.awt.Color.black);
    JLabel1.setBounds(15,162,470,24);
    getContentPane().add(connectionComboBox);
    connectionComboBox.setBounds(115,60,270,24);
    itemScrollPane.setOpaque(true);
    getContentPane().add(itemScrollPane);
    itemScrollPane.setBounds(15,190,470,200);
    itemScrollPane.getViewport().add(itemList);
    itemList.setBounds(0,0,467,197);
    exitButton.setText("Exit");
    exitButton.setActionCommand("Cancel");
    getContentPane().add(exitButton);
    exitButton.setBounds(212,400,75,24);
    connectButton.setText("Connect");
    connectButton.setActionCommand("jbutton");
    getContentPane().add(connectButton);
    connectButton.setBounds(400,120,85,24);
    titleLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
    titleLabel.setText("jlabel");
    getContentPane().add(titleLabel);
    titleLabel.setForeground(java.awt.Color.black);
    titleLabel.setFont(new Font("Dialog", Font.BOLD, 18));
    titleLabel.setBounds(25,10,450,30);
    selectedTextField.setEditable(false);
    getContentPane().add(selectedTextField);
    selectedTextField.setBackground(java.awt.Color.white);
    selectedTextField.setBounds(16,120,384,24);
    //}}

    //{{INIT_MENUS

    //}}

    //{{REGISTER_LISTENERS

    connectionComboBox.addItemListener(this);

    connectButton.addActionListener(this);
    exitButton.addActionListener(this);

    itemList.addMouseListener(this);
    //}}

    itemList.setModel(listModel);
    connectionComboBox.addItem(LINKS_TITLE);
    for( int i=0; i<LINK_FILE_NAMES.length; i++){
        connectionComboBox.addItem(LINK_FILE_NAMES[i]);
    }
}

```

```

    }
}

/**
 * default constructor
 */
public ReviewComponentContentDialog() {
    this((Frame)null);
}

/**
 * After select a step and press OK button from ListDialog, it will be launched
 * @param String and Vector[]
 */
public ReviewComponentContentDialog(String title, Vector[] storedVector){
    this();
    this.titleLabel.setText(title);
    this.storedVector = storedVector;
}

/**
 * yet another constructor with only the dialog window title changed to sTitle
 * @param String
 */
public ReviewComponentContentDialog(String sTitle) {
    this();
    setTitle(sTitle);
}

/**
 * a method to control the dialogs visibility based upon b
 * @param boolean
 */
public void setVisible(boolean b) {
    if (b)
        setLocation(50, 50);
    super.setVisible(b);
}

/**
 * the main procedure for unit testing
 */
static public void main(String[] args) {
    (new ReviewComponentContentDialog()).setVisible(true);
}

public void addNotify()
{
    // Record the size of the window prior to calling parents addNotify.
    Dimension size = getSize();

    super.addNotify();

    if (frameSizeAdjusted)
        return;
    frameSizeAdjusted = true;

    // Adjust size of frame according to the insets
    Insets insets = getInsets();
    setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height);
}

// Used by addNotify
boolean frameSizeAdjusted = false;

//{{DECLARE_CONTROLS
javax.swing.JLabel JLabel1 = new javax.swing.JLabel();
javax.swing.JComboBox connectionComboBox = new javax.swing.JComboBox();

```

```

        javax.swing.JScrollPane itemScrollPane = new javax.swing.JScrollPane();
        javax.swing.JList itemList = new javax.swing.JList();
        javax.swing.JButton exitButton = new javax.swing.JButton();
        javax.swing.JButton connectButton = new javax.swing.JButton();
        javax.swing.JLabel titleLabel = new javax.swing.JLabel();
        javax.swing.JTextField selectedTextField = new javax.swing.JTextField();
    //}}

/**
 * a method for handling item state changes for connection combo box
 */
    public void itemStateChanged(java.awt.event.ItemEvent event) {
        Object object = event.getSource();
        if (object == connectionComboBox)
            connectionComboBox_itemStateChanged(event);
    }

/**
 * Select a link type
 */
    void connectionComboBox_itemStateChanged(java.awt.event.ItemEvent event) {
        if( event.getStateChange() == event.SELECTED ){
            selectedTextField.setText("");
            listModel.removeAllElements();
            String selectedItem = (String)event.getItem();
            for( int i=0; i<LINK_FILE_NAMES.length; i++){
                if( selectedItem.equals(LINK_FILE_NAMES[i]) ){
                    for( int j=0; j<storedVector[i].size(); j++ ){
                        listModel.addElement(storedVector[i].elementAt(j));
                    }
                    i= LINK_FILE_NAMES.length;
                }
            }
        }
    }

/**
 * method to handle standard action events
 */
    public void actionPerformed(java.awt.event.ActionEvent event){
        Object object = event.getSource();
        if (object == connectButton) // connect button pressed
            connectButton_actionPerformed(event);
        else if (object == exitButton) // exit button pressed
            exitButton_actionPerformed(event);
    }

/**
 * Connect to an application which matches with selected item from connectionComboBox
 * and view the file which the name is the content of selectedTextField link
 */
    void connectButton_actionPerformed(java.awt.event.ActionEvent event)
    {
        String s = " " + selectedTextField.getText();
        if( !s.equals("") ){
            if( connectionComboBox.getSelectedIndex()>0 ){
                int index = connectionComboBox.getSelectedIndex()-1;
                if( index == 3 ){
                    (new PersonnelFrame(s.trim(), "View")).setVisible(true);
                }
            }
            else{
                try {
                    Runtime runtime = Runtime.getRuntime();
                    runtime.exec(EXECUTIONS[index]+s);
                }
                catch( Exception e ) {
                    throw new RuntimeException(e.toString());
                }
            }
        }
    }

```

```

    }
    }
}

/**
 * Exit ReviewComponentContentDialog
 */
void exitButton_actionPerformed(java.awt.event.ActionEvent event) {
    setVisible( false );
    dispose();
}

// standard mouse events for future functionality
public void mouseEntered(MouseEvent event) { }
public void mouseExited(MouseEvent event) { }
public void mouseReleased(MouseEvent event) { }
public void mousePressed(MouseEvent event) { }

/**
 * method to handle mouse clicked event
 */
public void mouseClicked(MouseEvent event) {
    Object object = event.getSource();
    if (object == itemList)
        itemList_mouseClicked(event);
}

/**
 * Set the selectedTextField when a user selects an item from this list
 */
void itemList_mouseClicked(java.awt.event.MouseEvent event)
{
    if( event.getClickCount() > 0) {
        String s = (String)this.itemList.getSelectedValue();
        this.selectedTextField.setText(s);
    }
}
}

```

19. Cases.SkillTableFrame

```

/**-----
 * @Filename: SkillTable.java
 * @Date: 3-7-2003
 * @Author: Le Hahn but modified by Arthur Clomera for NPS Thesis
 * Description: a table of skill IDs, skill names, and skill levels
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_Cases
 * Modified with standard event handlers and listeners
 * Removed dependencies upon Visual Cafe
 * @Compiler: JDK 1.3.1
 * -----
 */
package Cases;

import java.awt.*;
import java.awt.event.ActionListener;
import java.util.Vector;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;

```

```

////////////////////////////////////

```

```

/**
 * SkillTableFrame : a table of skill IDs, skill names, and skill levels
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_Cases
 */
////////////////////////////////////
public class SkillTableFrame extends javax.swing.JFrame implements CasesTitle, ActionListener
{
    /**
     * scf : parent frame to launch this frame
     */
    StepContentFrame scf = null;

    /**
     * pf : parent frame to launch this frame
     */
    PersonnelFrame pf = null;

    /**
     * skillVector : selected skill to return to parent frame
     */
    private Vector skillVector = new Vector();

    /**
     * skillModel : monitor skill table
     */
    protected DefaultTableModel skillModel = null;

    /**
     * Build SkillTableFrame
     */
    public SkillTableFrame() {
        initGUI();
    }

    /**
     * method to initialize the GUI and its components
     */
    private void initGUI() {

        //{ {INIT_CONTROLS
        setTitle("Skill List");
        getContentPane().setLayout(null);
        setSize(405,305);
        setVisible(false);
        tableScrollPane.setOpaque(true);
        getContentPane().add(tableScrollPane);
        tableScrollPane.setBounds(33,54,324,169);
        tableScrollPane.getViewport().add(skillTable);
        skillTable.setBounds(0,0,321,166);
        JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        JLabel1.setText("Skill List");
        getContentPane().add(JLabel1);
        JLabel1.setForeground(java.awt.Color.black);
        JLabel1.setFont(new Font("Dialog", Font.BOLD, 16));
        JLabel1.setBounds(15,6,375,47);
        OKButton.setText("OK");
        OKButton.setActionCommand("OK");
        getContentPane().add(OKButton);
        OKButton.setBounds(121,252,73,24);
        cancelButton.setText("Cancel");
        cancelButton.setActionCommand("Cancel");
        getContentPane().add(cancelButton);
        cancelButton.setBounds(211,252,73,24);
        //}}

        //{ {INIT_MENUS

```

```

        //}}

        //{{REGISTER_LISTENERS

        OKButton.addActionListener(this);
        cancelButton.addActionListener(this);
        //}}

    /**
     * create the new skill table
     */
        createSkillTable();
    }

    /**
     * StepContentFrame launch this frame when skill button is slected
     */
    public SkillTableFrame(StepContentFrame scf){
        this();
        this.scf = scf;
    }

    /**
     * PersonnelFrame launch this frame when skill button is slected
     */
    public SkillTableFrame(PersonnelFrame pf){
        this();
        this.pf = pf;
    }

    /**
     * construct to set the frame title to sTitle
     * @param String
     */
        public SkillTableFrame(String sTitle)    {
            this();
            setTitle(sTitle);
        }

    /**
     * this method controls the visibility of the frame based upon b
     * @param boolean
     */
        public void setVisible(boolean b) {
            if (b)
                setLocation(50, 50);
            super.setVisible(b);
        }

    /**
     * a main procedure for unit testing
     */
        static public void main(String[] args)    {
            (new SkillTableFrame()).setVisible(true);
        }

    /**
     * overrides the super addNotify() method
     */
        public void addNotify()    {
            // Record the size of the window prior to calling parents addNotify.
            Dimension size = getSize();

            super.addNotify();

            if (frameSizeAdjusted)
                return;
            frameSizeAdjusted = true;
        }

```

```

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    //{DECLARE_CONTROLS
    javax.swing.JScrollPane tableScrollPane = new javax.swing.JScrollPane();
    javax.swing.JTable skillTable = new javax.swing.JTable();
    javax.swing.JLabel JLabel1 = new javax.swing.JLabel();
    javax.swing.JButton OKButton = new javax.swing.JButton();
    javax.swing.JButton cancelButton = new javax.swing.JButton();
    //}}

    //{DECLARE_MENUS
    //}}

/**
 * a method to handle standard action events
 */
    public void actionPerformed(java.awt.event.ActionEvent event) {
        Object object = event.getSource();
        if (object == OKButton) // okay button pressed
            OKButton_actionPerformed(event);
        else if (object == cancelButton) // cancel button pressed
            cancelButton_actionPerformed(event);
    }

/**
 * Return all selected skills to a parent frame and exit SkillTableFrame
 */
    void OKButton_actionPerformed(java.awt.event.ActionEvent event)
    {
        int[] selectedRows = skillTable.getSelectedRows();
        for( int i=0; i<selectedRows.length; i++){
            int index = selectedRows[i];
            String s = skillModel.getValueAt(index,0)+" : "+skillModel.getValueAt(index,1)+" : "+skillModel.getValueAt(index,2);
            skillVector.addElement(s);
        }
        if( this.scf != null ){
            this.scf.setSkillComboBox(skillVector);
        }
        else if( this.pf != null ){
            this.pf.setSkillComboBox(skillVector);
        }
        setVisible(false);
        dispose();
    }

/**
 * Exit this frame without return anything
 */
    void cancelButton_actionPerformed(java.awt.event.ActionEvent event)
    {
        setVisible(false);
        dispose();
    }

/**
 * Create a skill table with 3 columns and 21 rows
 */

```

```

void createSkillTable(){
    String[] skillColumnNames = {"Skill ID", "Skill Name", "Skill Level" };
    final Object[][] skillData = new Object[20][3];

    for( int i = 1; i <SKILL_ID; ++i ){
        skillData[i-1][0] = ""+i;
    }

    for( int j=0; j<SKILL_LIST.length; j++ ){
        skillData[j][1] = SKILL_LIST[j];
        skillData[j][2]=""+0;
    }
    skillModel = new DefaultTableModel(skillData, skillColumnNames);
    JComboBox skillLevelComboBox = new JComboBox();
    for( int k= 0; k <SKILL_LEVEL; ++k ){
        skillLevelComboBox.addItem(""+k);
    }
    skillTable.setModel(skillModel);

    TableColumn skillLevelColumn = skillTable.getColumn(skillColumnNames[2]);

    // Use the combo box as the editor in the "Skill Level" column.
    skillLevelColumn.setCellEditor(new DefaultCellEditor(skillLevelComboBox));
}
}

```

20. Cases.StepContent

```

/**-----
 * @Filename: StepContent.java
 * @Date: 3-7-2003
 * @Author: Le Hahn but modified by Arthur Clomera for NPS Thesis
 * Description: Create a step content object and save it in step.cnt file
 *
 * Implement observable for MVC pattern
 * @Compiler: JDK 1.3.1
 * -----
 */
package Cases;

import java.io.Serializable;
import java.util.Observable;
import java.util.Vector;
/**
 * Step Content Object which is used to save in step.cnt file
 */

////////////////////////////////////
/**
 * StepContent : Create a step content object and save it in step.cnt file
 */
////////////////////////////////////
public class StepContent extends Observable implements Serializable{

    /**
     * stepName : step version
     */
    private String stepName = null;

    /**
     * status : status of the step, eg. approved, scheduled,complete,...
     */
    private String status = null;

    /**

```



```

    * skill : a vector of skills
    */
private Vector skill = new Vector();

/**
 * securityLevel : security level of the step
 */
private int securityLevel = 0;

/**
 * organizer : a person's ID to organize the step
 */
private String organizer = null;

/**
 * predecessors : a vector of atomics
 */
private Vector predecessors = new Vector();

/**
 * priority : priority of the step
 */
private int priority = 0;

/**
 * estimateDuration : estimate how long the job will finish
 */
private int estimatedDuration = 0;

/**
 * deadline : deadline of the job
 */
private String deadline = null;

/**
 * earliestStartTime : the time to start in the plan
 */
private String earliestStartTime = null;

/**
 * finishTime : when the job is finished
 */
private String finishTime = null;

/**
 * realStartTime : the actual day to start the job
 */
private String realStartTime = null;

/**
 * manager : a person's ID to manage the job
 */
private String manager = null;

/**
 * evaluation : description of an evaluation
 */
private String evaluation = null;

/**
 * evaluator : a person's ID to evaluate the job
 */
private String evaluator = null;

/**
 * This StepContent constructor is used to create a step content object
 */
public StepContent( String stepName, String status, Vector skill,

```

```

        int securityLevel, String evaluation, String evaluator,
        String organizer, Vector predecessors, int priority,
        int estimatedDuration, String deadline, String startTime,
        String finishTime, String manager){
    this.stepName = stepName;
    this.status = status;
    this.skill = skill;
    this.securityLevel = securityLevel;
    this.evaluation = evaluation;
    this.evaluator = evaluator;
    this.organizer = organizer;
    this.predecessors = predecessors;
    this.priority = priority;
    this.estimatedDuration = estimatedDuration;
    this.deadline = deadline;
    this.earliestStartTime = startTime;
    this.finishTime = finishTime;
    this.manager = manager;
}

/**
 * Empty StepContent constructor
 */
public StepContent(){
}

/**
 * Set a name for the step
 *
 * @param s : name of the step
 */
public void setStepName(String s){
    this.stepName = s;
}

/**
 * Get the step's name
 *
 * @return stepName : the step's name
 */
public String getStepName(){
    return this.stepName;
}

/**
 * Set status for the step
 *
 * @param s : status of the step
 */
public void setStatus(String s){
    this.status = s;
}

/**
 * Status of the step
 *
 * @return status : status of the step
 */
public String getStatus(){
    return this.status;
}

/**
 * Set skills for the step
 *
 * @param v : vector of skills
 */
public void setSkill(Vector v){

```

```

        this.skill = v;
    }

    /**
     * Skills of the step
     *
     * @return skill : vector of skills of the step
     */
    public Vector getSkill(){
        return skill;
    }

    /**
     * Set security level for the step
     *
     * @param i: security level
     */
    public void setSecurityLevel(int i){
        this.securityLevel = i;
    }

    /**
     * Security level of the step
     *
     * @return securityLevel : security level of the step
     */
    public int getSecurityLevel(){
        return this.securityLevel;
    }

    /**
     * Set evaluation of the step
     *
     * @param s : a string of evaluation
     */
    public void setEvaluation(String s){
        this.evaluation = s;
    }

    /**
     * Evaluation of the step
     *
     * @return evaluation : evaluation of the step
     */
    public String getEvaluation(){
        return this.evaluation;
    }

    /**
     * Set evaluator for the step
     *
     * @param s : evaluator's name
     */
    public void setEvaluator(String s){
        this.evaluator = s;
    }

    /**
     * Evaluator of the step
     *
     * @return evaluator : evaluator of the step
     */
    public String getEvaluator(){
        return this.evaluator;
    }

    /**
     * Set organizer for the step

```

```

*
* @param s : the name of organizer
*/
public void setOrganizer(String s){
    this.organizer = s;
}

/**
* Organizer of the step
*
* @return organizer : organizer of the step
*/
public String getOrganizer(){
    return this.organizer;
}

/**
* Set predecessors for the step
*
* @param v : vector of atomics
*/
public void setPredecessors(Vector v){
    this.predecessors = v;
}

/**
* Predecessors of the step
*
* @return predecessors : a vector of atomics
*/
public Vector getPredecessors(){
    return this.predecessors;
}

/**
* Set priority for the step
*
* @param i : level of priority
*/
public void setPriority(int i){
    this.priority = i;
}

/**
* Priority of the step
*
* @return priority : an integer of priority level
*/
public int getPriority(){
    return this.priority;
}

/**
* Set duration for the step
*
* @param i : prediction of how long it will take to finish the job
*/
public void setDuration(int i){
    this.estimatedDuration = i;
}

/**
* Estimate duration to finish the job
*
* @return estimatedDuration : time to finish the job
*/
public int getDuration(){
    return this.estimatedDuration;
}

```

```

    }

    /**
     * Set deadline of the job
     *
     * @param d : exactly day to be done this job
     */
    public void setDeadline(String d){
        this.deadline = d;
    }

    /**
     * Deadline of the job
     *
     * @return deadline : a string to express the deadline
     */
    public String getDeadline(){
        return this.deadline;
    }

    /**
     * Set earliest start time for the job
     *
     * @param d : a string to express the earliest start time
     */
    public void setStartTime(String d){
        this.earliestStartTime = d;
    }

    /**
     * The day to plan starting the job
     *
     * @return earliestStartTime : plan to start on this day
     */
    public String getStartTime(){
        return this.earliestStartTime;
    }

    /**
     * Set the finish time for the job
     *
     * @param d : a string to express the finish day
     */
    public void setFinishTime(String d){
        this.finishTime = d;
    }

    /**
     * The finish day
     *
     * @return finishTime : a string to express the finish day
     */
    public String getFinishTime(){
        return this.finishTime;
    }

    /**
     * Set the exactly day to start the job
     *
     * @param realStartTime : a string to express the day to start the job
     */
    public void setRealStartTime(String d){
        this.realStartTime = d;
    }

    /**
     * The day to start the job
     *

```

```

    * @return realStartTime : a string to express the day to start the job
    */
    public String getRealStartTime(){
        return this.realStartTime;
    }

    /**
     * Set manager's ID for the step
     *
     * @param s : the manager's ID
     */
    public void setManager(String s){
        this.manager = s;
    }

    /**
     * The manager's ID to manage the job
     *
     * @return manager : the manager's ID
     */
    public String getManager(){
        return this.manager;
    }
}

```

21. Cases.StepContentFrame

```

/**-----
 * @Filename: StepContentFrame.java
 * @Date: 3-7-2003
 * @Author: Le Hahn but modified by Arthur Clomera for NPS Thesis
 * Description: Use to create/delete/view step content of the selected step
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_StepContent
 * Modified with GUI Calendar subsystem, standard event handlers and listeners
 * Implement observable for MVC pattern
 * removed dependencies on Visual Cafe
 * @Compiler: JDK 1.3.1
 * -----
 */
package Cases;

import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.io.*;
import java.text.DateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.StringTokenizer;
import java.util.Vector;
import javax.swing.*;
import Cases.GUI.CalendarDialog.DateButton;
import Cases.GUI.CalendarDialog.DateChooser;
import Cases.Interfaces.I_StepContent;

////////////////////////////////////
/**
 * StepContentFrame : Use to create/delete/view step content of the selected step
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_StepContent
 */
////////////////////////////////////
public class StepContentFrame extends javax.swing.JFrame implements CasesTitle, I_StepContent, ActionListener

```

```

{
    public MyTextField estDurationTextField = new MyTextField();
    public String stepName = "";
    public CasesFrame casesFrame = null;
    /**
     * atomicsVector : contains all atomics of the selected step
     */
    public Vector atomicsVector = new Vector();

    /**
     * selectedSkill : contains all selected skills from SkillTableFrame
     */
    public Vector selectedSkill = null;

    /**
     * selectedPred : contains all selected predecessors from ListDialog
     */
    public Vector selectedPred = null;

    /**
     * pathName : the complete path of current step
     */
    public String pathName = null;

    /**
     * Build StepContentFrame
     */
    public StepContentFrame() {
        initGUI();
    }

    /**
     * method for initializing GUI and its components
     */
    private void initGUI()
    {
        //{ INIT_CONTROLS

        label3 = new Label();
        label2 = new Label();
        label1 = new Label();
        setTitle("SPIDER-Step Content");
        getContentPane().setLayout(null);
        setSize(700,550);
        setVisible(false);
        getContentPane().add(saveButton);
        saveButton.setBounds(0,0,0,0);
        getContentPane().add(deleteButton);
        deleteButton.setBounds(0,0,0,0);
        exitButton.setText("Exit");
        exitButton.setActionCommand("Save");
        getContentPane().add(exitButton);
        exitButton.setBounds(590,470,75,24);
        titleLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        titleLabel.setText("Step Content:");
        getContentPane().add(titleLabel);
        titleLabel.setForeground(java.awt.Color.black);
        titleLabel.setFont(new Font("Dialog", Font.BOLD, 18));
        titleLabel.setBounds(82,30,130,30); //y=6
        /**stepVersionTF**/
        stepVersionTextField.setEditable(false);
        getContentPane().add(stepVersionTextField);
        stepVersionTextField.setBackground(java.awt.Color.white);
        stepVersionTextField.setBounds(102,105,200,24);//246,55,300,24);
        /**
        /**stepVersionLabel**/
        JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel1.setText("Step Version");

```

```

getContentPane().add(JLabel1);
JLabel1.setForeground(java.awt.Color.black);
JLabel1.setBounds(1,105,100,24);//154,55,90,24);
/**/
getContentPane().add(managerComboBox);
managerComboBox.setBounds(472,405,200,24);
JLabel2.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel2.setText("Manager");
getContentPane().add(JLabel2);
JLabel2.setForeground(java.awt.Color.black);
JLabel2.setBounds(371,405,100,24);
/**Status label**/
JLabel3.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel3.setText("Status");
getContentPane().add(JLabel3);
JLabel3.setForeground(java.awt.Color.black);
JLabel3.setBounds(1,155,100,24);//1,105,100,24);
getContentPane().add(statusComboBox);
statusComboBox.setBounds(102,155,200,24);//102,105,200,24);
JLabel5.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel5.setText("Evaluation");
getContentPane().add(JLabel5);
JLabel5.setForeground(java.awt.Color.black);
JLabel5.setBounds(1,305,100,24);
JLabel6.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel6.setText("Evaluator");
getContentPane().add(JLabel6);
JLabel6.setForeground(java.awt.Color.black);
JLabel6.setBounds(1,355,100,24);
getContentPane().add(evaluatorComboBox);
evaluatorComboBox.setBounds(102,355,200,24);
JLabel7.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel7.setText("Security Level");
getContentPane().add(JLabel7);
JLabel7.setForeground(java.awt.Color.black);
JLabel7.setBounds(1,255,100,24);
getContentPane().add(securityLevelComboBox);
securityLevelComboBox.setBounds(102,255,200,24);
JLabel9.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel9.setText("Organizer");
getContentPane().add(JLabel9);
JLabel9.setForeground(java.awt.Color.black);
JLabel9.setBounds(1,405,100,24);
getContentPane().add(organizerComboBox);
organizerComboBox.setBounds(101,405,200,24);
getContentPane().add(predecessorsComboBox);
predecessorsComboBox.setBounds(472,105,200,24);
JLabel12.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel12.setText("Priority");
getContentPane().add(JLabel12);
JLabel12.setForeground(java.awt.Color.black);
JLabel12.setBounds(371,155,100,24);
getContentPane().add(priorityComboBox);
priorityComboBox.setBounds(472,155,200,24);
/**SkillComboBox
getContentPane().add(skillComboBox);
skillComboBox.setBounds(102,205,200,24);//102,155,200,24);
/**/
JLabel16.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel16.setText("Estimated Duration");
getContentPane().add(JLabel16);
JLabel16.setForeground(java.awt.Color.black);
JLabel16.setBounds(361,205,110,24);
getContentPane().add(evaluationTextField);
evaluationTextField.setBounds(101,305,200,24);
selectedLabel.setText("JLabel");
getContentPane().add(selectedLabel);
selectedLabel.setForeground(java.awt.Color.black);

```



```

        selectedLabel.setFont(new Font("Dialog", Font.BOLD, 14));
        selectedLabel.setBounds(217,30,400,30); //y=6
        predecessorsButton.setText("Predecessors");
        predecessorsButton.setActionCommand("jbutton");
        getContentPane().add(predecessorsButton);
        predecessorsButton.setBounds(351,105,120,24);
        /**SkillButton
        skillButton.setText("Skill");
        skillButton.setActionCommand("Skill");
        getContentPane().add(skillButton);
        skillButton.setBounds(21,205,80,24);//21,155,80,24);
        /**/
        deadlineButton.setText("Deadline");
        deadlineButton.setActionCommand("Deadline");
        getContentPane().add(deadlineButton);
        deadlineButton.setBounds(new java.awt.Rectangle(471, 249, 198, 29));
        startTimeButton.setText("Earliest Start Time");
        startTimeButton.setActionCommand("Earliest Start Time");
        getContentPane().add(startTimeButton);
        startTimeButton.setBounds(new java.awt.Rectangle(472, 305, 196, 24));
        finishTimeButton.setText("Finish Time");
        finishTimeButton.setActionCommand("Finish Time");
        getContentPane().add(finishTimeButton);
        finishTimeButton.setBounds(new java.awt.Rectangle(472, 355, 198, 24));
        //} }

        //{{INIT_MENU
        //}}

        //{{REGISTER_LISTENERS
        saveButton.addActionListener(this);
        deleteButton.addActionListener(this);
        exitButton.addActionListener(this);
        predecessorsButton.addActionListener(this);
        skillButton.addActionListener(this);
        deadlineButton.addActionListener(this);
        startTimeButton.addActionListener(this);
        finishTimeButton.addActionListener(this);
        //}}

//Set status combobox
for( int k=0; k<STATUS.length; k++ ){
    statusComboBox.addItem(STATUS[k]);
}

//Set security level from 0..5
for( int i=0; i<SECURITY_LEVEL; i++){
    securityLevelComboBox.addItem(i+"");
}

//Set priority combobox from 0..5
for( int j=0; j<PRIORITY_LEVEL; j++){
    priorityComboBox.addItem(j+"");
}

        getContentPane().add(estDurationTextField);
        getContentPane().add(label1);
        getContentPane().add(label2);
        getContentPane().add(label3);
        estDurationTextField.setBounds(472,205,200,24);
        label1.setText("Deadline");
        label1.setBounds(new java.awt.Rectangle(357, 251, 107, 27));
        label2.setText("Earliest Start Time");
        label2.setBounds(new java.awt.Rectangle(357, 304, 106, 23));
        label3.setText("Finish Time");
        label3.setBounds(new java.awt.Rectangle(356, 357, 109, 20));
    }

/**

```

```

* CasesFrame launch this frame when Step Content menu item is selected or
* Step Content button from Trace Frame is pressed
*
* @param traceFrame : = null if launched from Step Content menu item
*                   != null if launched from Step Content button
* @param pathName : the path of current step
* @param atomics : vector of the current step's atomics
*/
public StepContentFrame(CasesFrame casesFrame, TraceFrame traceFrame, String stepName, String pathName, Vector atomics)
{
    this();
    this.casesFrame = casesFrame;
    this.atomicsVector = atomics;
    this.selectedLabel.setText( pathName );
    this.stepVersionTextField.setText( stepName );
    this.stepName = stepName;
    this.pathName = pathName;
    setStakeholders();
    Calendar theCalendar = Calendar.getInstance();

    try{
        File f = new File(pathName+"\\step.cnt");
        if( f.exists() ){
            FileInputStream fileInput = new FileInputStream(f);
            ObjectInputStream oi = new ObjectInputStream( fileInput );
            if( oi != null ){
                setInitial((StepContent)oi.readObject());
            }
            else{
                try{
                    DateFormat df = DateFormat.getDateInstance();
                    Date d = theCalendar.getTime();
                    d = deadlineButton.getDate();
                }
                catch(Exception e){System.out.println(e);}
            }
            oi.close();
            fileInput.close();
        }

        catch( IOException io ){
            debug("IOException: "+io);
            try{
                DateFormat df = DateFormat.getDateInstance();
                Date d = theCalendar.getTime();
            }
            catch( Exception e ){System.out.println(e);}

            }
        catch( ClassNotFoundException c ){
            debug("ClassNotFoundException: "+c);
        }
        if( traceFrame != null ){
            setReadOnly();
        }
        else{
            saveButton.setText("Save");
            saveButton.setActionCommand("Save");
            getContentPane().add(saveButton);
            saveButton.setBounds(440,470,75,24);
            deleteButton.setText("Delete");
            deleteButton.setActionCommand("Save");
            getContentPane().add(deleteButton);
            deleteButton.setBounds(515,470,75,24);
        }
    }

}
/**

```

```

* constructor to set title to sTitle
* @param String
*/
    public StepContentFrame(String sTitle)    {
        this();
        setTitle(sTitle);
    }

/**
* method to control frame visibility based upon b
* @param boolean
*/
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

/**
* main procedure for unit testing
*/
    static public void main(String[] args)    {
        (new StepContentFrame()).setVisible(true);
    }

/**
* overrides the super addNotify() method
*/
    public void addNotify()    {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

// Used by addNotify
boolean frameSizeAdjusted = false;

//{{DECLARE_CONTROLS
javax.swing.JButton saveButton = new javax.swing.JButton();
javax.swing.JButton deleteButton = new javax.swing.JButton();
javax.swing.JButton exitButton = new javax.swing.JButton();
javax.swing.JLabel titleLabel = new javax.swing.JLabel();
javax.swing.JTextField stepVersionTextField = new javax.swing.JTextField();
javax.swing.JLabel JLabel1 = new javax.swing.JLabel();
javax.swing.JComboBox managerComboBox = new javax.swing.JComboBox();
javax.swing.JLabel JLabel2 = new javax.swing.JLabel();
javax.swing.JLabel JLabel3 = new javax.swing.JLabel();
javax.swing.JComboBox statusComboBox = new javax.swing.JComboBox();
javax.swing.JLabel JLabel4 = new javax.swing.JLabel();
javax.swing.JLabel JLabel5 = new javax.swing.JLabel();
javax.swing.JLabel JLabel6 = new javax.swing.JLabel();
javax.swing.JComboBox evaluatorComboBox = new javax.swing.JComboBox();
javax.swing.JLabel JLabel7 = new javax.swing.JLabel();
javax.swing.JComboBox securityLevelComboBox = new javax.swing.JComboBox();
javax.swing.JLabel JLabel9 = new javax.swing.JLabel();

```

```

    javax.swing.JComboBox organizerComboBox = new javax.swing.JComboBox();
    javax.swing.JComboBox predecessorsComboBox = new javax.swing.JComboBox();
    javax.swing.JLabel JLabel12 = new javax.swing.JLabel();
    javax.swing.JComboBox priorityComboBox = new javax.swing.JComboBox();
    javax.swing.JComboBox skillComboBox = new javax.swing.JComboBox();
    javax.swing.JLabel JLabel16 = new javax.swing.JLabel();
    javax.swing.JTextField evaluationTextField = new javax.swing.JTextField();
    javax.swing.JLabel selectedLabel = new javax.swing.JLabel();
    javax.swing.JButton predecessorsButton = new javax.swing.JButton();
    javax.swing.JButton skillButton = new javax.swing.JButton();
    DateButton deadlineButton = new DateButton();
    DateButton startTimeButton = new DateButton();
    DateButton finishTimeButton = new DateButton();
public Label label1;
public Label label2;
public Label label3;
    //javax.swing.JTextField estDurationTextField = new javax.swing.JTextField();
    //}}

    //{{DECLARE_MENUS
    //}}

/**
 * method to handle standard action events
 */
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == saveButton) // save button pressed
            saveButton_actionPerformed(event);
        else if (object == deleteButton) // delete button pressed
            deleteButton_actionPerformed(event);
        else if (object == exitButton) // exit button pressed
            exitButton_actionPerformed(event);
        else if (object == predecessorsButton) // predecessor button pressed
            predecessorsButton_actionPerformed(event);
        else if (object == skillButton) // skill button pressed
            skillButton_actionPerformed(event);
        else if (object == deadlineButton) // deadline button pressed
            deadlineButton_actionPerformed(event);
        else if (object == startTimeButton) // start time button pressed
            startTimeButton_actionPerformed(event);
        else if (object == finishTimeButton) // finish time button pressed
            finishTimeButton_actionPerformed(event);
    }

/**
 * method to check personnel objects
 */
    public void checkPersonnelObjects(){
        String status = (String)this.statusComboBox.getSelectedItem();
        Vector updatePersonnels = new Vector();
        Vector majorJobs = null;
        Vector minorJobs = null;
        StepContent stepContent = null;
        Personnel personnel = null;
        String s = "";
        if( !status.equals("Assigned") ){
            Vector personnelVector = (Vector)this.casesFrame.getPersonnelVector();
            if( personnelVector != null ){
                for( int i=0; i<personnelVector.size(); i++ ){
                    personnel = (Personnel)personnelVector.elementAt(i);
                    majorJobs = (Vector)personnel.getMajorJobs();
                    minorJobs = (Vector)personnel.getMinorJobs();
                    if( majorJobs.size() > 0 ){
                        for( int j=0; j<majorJobs.size(); j++ ){
                            stepContent = (StepContent)majorJobs.elementAt(j);
                            s = (String)stepContent.getStepName();

```

```

        if( s.equals(this.stepName) ){
            majorJobs.removeElementAt(j);
        }
    }
}

if( minorJobs.size() > 0 ){
    for( int k=0; k<minorJobs.size(); k++ ){
        stepContent = (StepContent)minorJobs.elementAt(k);
        s = (String)stepContent.getStepName();
        if( s.equals(this.stepName) ){
            minorJobs.removeElementAt(k);
        }
    }
}
personnel.setMajorJobs(majorJobs);
personnel.setMinorJobs(minorJobs);
updatePersonnels.addElement(personnel);
}
}

if( updatePersonnels.size() > 0 ){
    this.casesFrame.savePersonnelVector(updatePersonnels);
}
}

/**
 * Create a step content object and save it in step.cnt file
 * under the current step path. Then exit this frame
 */
public void saveButton_actionPerformed(java.awt.event.ActionEvent event)
{
    try{
        FileOutputStream fileOutput = new FileOutputStream(this.pathName+"\\step.cnt");
        ObjectOutputStream oo = new ObjectOutputStream(fileOutput);
        if( oo != null ){
            oo.writeObject((StepContent)getStepContent());
        }
        oo.flush();
        oo.close();
        fileOutput.close();
        checkPersonnelObjects();
    }
    catch(IOException io){
        debug("IOException: "+io);
    }
    exitButton_actionPerformed(event);
}

/**
 * Confirm message will ask a user before agree to delete step.cnt file
 */
public void deleteButton_actionPerformed(java.awt.event.ActionEvent event)
{
    {
        File f = new File(this.pathName, "step.cnt");
        if( f.exists() ){
            int result = JOptionPane.showConfirmDialog( this, "step.cnt file will be deleted. Would you like to continue?",
                "Confirm Message",JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
            //result = 0 ==> yes
            //result = 1 ==> no
            if( result == 0 ){
                f.delete();
                exitButton_actionPerformed(event);
            }
        }
        else{
            JOptionPane.showMessageDialog(this, "step.cnt file does not exist!",
                "Warning Message",JOptionPane.INFORMATION_MESSAGE);
        }
    }
}

```

```

    }
}

/**
 * Exit this frame
 */
public void exitButton_actionPerformed(java.awt.event.ActionEvent event)
{
    setVisible( false );
    dispose();
}

/**
 * Set initial of this frame if the selected step already created step.cnt file
 *
 * @param stepContent : step content object of this current step
 */
public void setInitial( StepContent stepContent ){
    Vector temp = new Vector();

    this.stepVersionTextField.setText((String)stepContent.getStepName());
    this.statusComboBox.setSelectedItem((String)stepContent.getStatus());

    //Set up skill comboBox
    temp = (Vector)stepContent.getSkill();
    if( temp != null ){
        this.selectedSkill = new Vector();
        this.skillComboBox.addItem(SKILL_TITLE);
        for( int i=0; i<temp.size(); i++ ){
            this.skillComboBox.addItem(temp.elementAt(i));
            this.selectedSkill.addElement(temp.elementAt(i));
        }
    }
    this.securityLevelComboBox.setSelectedItem(""+stepContent.getSecurityLevel());
    this.evaluationTextField.setText((String)stepContent.getEvaluation());
    this.evaluatorComboBox.setSelectedItem((String)stepContent.getEvaluator());
    this.organizerComboBox.setSelectedItem((String)stepContent.getOrganizer());

    //Set up predecessors comboBox
    temp = new Vector();
    temp = (Vector)stepContent.getPredecessors();
    if( temp != null ){
        this.selectedPred = new Vector();
        this.predecessorsComboBox.addItem("Current Selected List");
        for( int i=0; i<temp.size(); i++ ){
            this.predecessorsComboBox.addItem(temp.elementAt(i));
            this.selectedPred.addElement(temp.elementAt(i));
        }
    }
    this.priorityComboBox.setSelectedItem(""+stepContent.getPriority());
    this.estDurationTextField.setText(""+stepContent.getDuration());
    this.deadlineButton.setLabel((String)stepContent.getDeadline());
    this.startTimeButton.setLabel((String)stepContent.getStartTime());
    this.finishTimeButton.setLabel((String)stepContent.getFinishTime());
    this.managerComboBox.setSelectedItem((String)stepContent.getManager());
}

/**
 * Create a step content object before save it
 *
 * @return stepContent object with all the information from this frame
 */
public StepContent getStepContent(){
    int security = (new Integer((String)this.securityLevelComboBox.getSelectedItem())).intValue();
    int priority = (new Integer((String)this.priorityComboBox.getSelectedItem())).intValue();
    String estDur = (String)this.estDurationTextField.getText();
    int duration = 0;
    if( !estDur.equals("") ){

```

```

        duration = (new Integer(estDur)).intValue();
    }
    StepContent stepContent = new StepContent();

    stepContent.setStepName((String)this.stepVersionTextField.getText());
    stepContent.setStatus((String)this.statusComboBox.getSelectedItem());
    stepContent.setSkill((Vector)this.selectedSkill);
    stepContent.setSecurityLevel(security);
    stepContent.setEvaluation((String)this.evaluationTextField.getText());
    stepContent.setEvaluator((String)this.evaluatorComboBox.getSelectedItem());
    stepContent.setOrganizer((String)this.organizerComboBox.getSelectedItem());
    stepContent.setPredecessors((Vector)this.selectedPred);
    stepContent.setPriority(priority);
    stepContent.setDuration(duration);
    stepContent.setDeadline((String)this.deadlineButton.getLabel());
    stepContent.setStartTime((String)this.startTimeButton.getLabel());
    stepContent.setFinishTime((String)this.finishTimeButton.getLabel());
    stepContent.setManager((String)this.managerComboBox.getSelectedItem());

    return stepContent;
}

/**
 * Get all personnel objects in stakeholder directory,
 * and add them in evaluator, organizer, and manager combo boxes
 */
public void setStakeHolders() {
    String[] list = (String[])STAKEHOLDER.list();
    if( list.length > 0 ){
        for( int i=0; i<list.length; i++ ){
            this.evaluatorComboBox.addItem(list[i]);
            this.organizerComboBox.addItem(list[i]);
            this.managerComboBox.addItem(list[i]);
        }
    }
}

/**
 * This function is called when SkillTableFrame's OK button is pressed,
 * it will return a vector of selected skills. Use this vector to set
 * skillComboBox
 *
 * @param v : vector of skills from SkillTableFrame
 */
public void setSkillComboBox(Vector v){
    this.selectedSkill = new Vector();
    this.skillComboBox.removeAllItems();
    if( v.size() > 0 ){
        this.skillComboBox.addItem(SKILL_TITLE);
        for( int i=0; i<v.size(); i++ ){
            this.skillComboBox.addItem(v.elementAt(i));
            this.selectedSkill.addElement(v.elementAt(i));
        }
    }
}

/**
 * Launch ListDialog to allows a user to select more than one atomics
 */
public void predecessorsButton_actionPerformed(java.awt.event.ActionEvent event)
{
    Vector v = new Vector();
    String parentPath = CASESDIRECTORY.getAbsolutePath();
    int length = parentPath.length();
    debug("Path length = "+length);
    for( int i=0; i<this.atomicsVector.size(); i++ ){
        File f = (File) this.atomicsVector.elementAt(i);
        String s = f.getAbsolutePath();

```

```

String sub1 = s.substring(length);
int index = sub1.indexOf("\\");
String sub2 = sub1.substring(index+1);

debug("Sub1 = "+sub1);
debug("Sub2 = "+sub2);
StringTokenizer st = new StringTokenizer(sub2, "\\");
String theAtomic = null;
int j=0;
while( st.hasMoreTokens()){
    String theString = st.nextToken();
    if( j==0 ){
        theAtomic = theString;
    }
    else if( j==1){
        theAtomic = theAtomic + theString;
    }
    else if( j==2){
        theAtomic = theAtomic + "-" + theString;
    }
    else if( j>2){
        theAtomic = theAtomic + "." + theString;
    }
    j++;
}
v.addElement(theAtomic);
}
(new ListDialog(this, "Predecessor List", v)).setVisible(true);
}

/**
 * Launch SkillTableFrame to allows a user to select more than one skills
 */
public void skillButton_actionPerformed(java.awt.event.ActionEvent event)
{
    (new SkillTableFrame(this)).setVisible(true);
}

/**
 * It is only called when Step Content button in TraceFrame is pressed.
 * The purpose is to view the content of this step, and the user won't allow
 * to create/edit/delete/save this step content at all.
 */
public void setReadOnly(){
    managerComboBox.setEnabled( false );
    statusComboBox.setEnabled( false );
    evaluatorComboBox.setEnabled( false );
    securityLevelComboBox.setEnabled( false );
    priorityComboBox.setEnabled( false );
    organizerComboBox.setEnabled( false );
    predecessorsButton.setEnabled( false );
    skillButton.setEnabled( false );
    evaluationTextField.setEditable( false );
    estDurationTextField.setEditable( false );

    evaluationTextField.setBackground(java.awt.Color.white);
    estDurationTextField.setBackground(java.awt.Color.white);

    startTimeButton.setEnabled( false );
    deadlineButton.setEnabled( false );
    finishTimeButton.setEnabled( false );
}

/**
 * Launch CalendarDialog
 */
public void startTimeButton_actionPerformed(java.awt.event.ActionEvent event)
{

```



```

        String s = this.startTimeButton.getLabel();
        if( !s.equals("") ){
            try{
                new DateChooser(this,s);
            }
            catch(Exception e){System.out.println(e);}
        }
    }
    else{
        new DateChooser(this,s);
    }
}

/**
 * Launch CalendarDialog
 */
public void deadlineButton_actionPerformed(java.awt.event.ActionEvent event)
{
    String s = this.deadlineButton.getLabel();
    if( !s.equals("") ){
        try{
            new DateChooser(this,s);
        }
        catch(Exception e){System.out.println(e);}
    }
    else{
        new DateChooser(this,s);
    }
}

/**
 * Launch CalendarDialog
 */
public void finishTimeButton_actionPerformed(java.awt.event.ActionEvent event)
{
    String s = this.finishTimeButton.getLabel();
    if( !s.equals("") ){
        try{
            new DateChooser(this,s);
        }
        catch(Exception e){System.out.println(e);}
    }
    else{
        new DateChooser(this,s);
    }
}

/**
 * Short cut to print the output
 * @param string : the output string
 */
void debug(String s){
    System.out.println(s);
}

/**
 * Custom JTextfield. It is used to create estDurationTextField since
 * it only allows a user to input integer. And, JTextfield doesn't have
 * that capability.
 */
private class MyTextField extends JTextfield
{
    protected void processKeyEvent( KeyEvent e )
    {
        if ( e.getModifiers() == 0 )
        {
            int keyChar = e.getKeyChar();
            if ( keyChar >= '0' && keyChar <= '9' ||
                keyChar == '\n' ){

```

```

        super.processKeyEvent( e );
    }
}
}

/**
 * Get an array of selected atomics from ListDialog and set predecessorsComboBox
 *
 * @param oa : an array of selected atomics, and elements of this array are File objects
 */
public void setPredecessorComboBox(Object[] oa){
    this.predecessorsComboBox.removeAllItems();
    this.selectedPred = new Vector();
    if( oa.length > 0 ){
        this.predecessorsComboBox.addItem("Current Selected List");
        for( int i=0; i<oa.length; i++ ){
            this.predecessorsComboBox.addItem(oa[i]);
            this.selectedPred.addElement(oa[i]);
        }
    }
}
}

```

22. Cases.StepType

```

/**-----
 * Filename: StepType.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: JDK 1.3.1
 * Description: Create a Step Type object and save it in component.cfg file
 * extended with QFD data and observable
 *-----
 */
package Cases;

import java.awt.*;
import java.io.Serializable;
import java.util.Hashtable;
import java.util.Observable;

/**
 * Step Type Object which is used to save in step.cfg file
 */

/**
 * StepType : Create a step type object and save it in step.cfg file
 */

public class StepType extends Observable implements Serializable {

    /**
     * stepID : step type ID
     */
    private String stepID = null;

    /**
     * stepName : name of the step type
     */
    private String stepName = null;

    /**
     * stepDescription : description of the step
     */

```

```

private String stepDescription = null;
/** secondary input component */
private int stepFrom;
    /** primary input component */
private int stepTo;
    /** steps position in point format */
    private Point stepPosition = new Point(0,0);

    /** hashtable to store qfd information */
private Hashtable stepQFDHashtable = new Hashtable();

/**
 * This StepType constructor is used to create step type object
 */
public StepType( String stepID, String stepName, String stepDescription, int stepFrom, int stepTo ){
    this.stepID = stepID;
    this.stepName = stepName;
    this.stepDescription = stepDescription;
    this.stepFrom = stepFrom;
    this.stepTo = stepTo;
}

/**
 * default constructor
 */
public StepType() {}

/**
 * Step type ID
 *
 * @param stepID : step type ID
 */
public String getStepID() {
    return this.stepID;
}

/** @param String newID */
public void setStepID(String newID) {
    this.stepID = newID;
    // must call setChanged before notifyObservers to
    // indicate model has changed
    setChanged();

    // notify Observers that model has changed
    notifyObservers();
}

/**
 * Step type name
 *
 * @param stepName : name of the step type
 */
public String getStepName(){ return this.stepName; }

/** @param String newName */
public void setStepName(String newName) {
    this.stepName = newName;
    // must call setChanged before notifyObservers to
    // indicate model has changed
    setChanged();

    // notify Observers that model has changed
    notifyObservers();
}

/**
 * Step type description
 */

```

```

    * @param stepDescription : the description of step type
    */
    public String getStepDescription(){ return this.stepDescription; }

    /** @param String new Description */
    public void setStepDescription (String newDescription) {
        this.stepDescription = newDescription;
        // must call setChanged before notifyObservers to
        // indicate model has changed
        setChanged();

        // notify Observers that model has changed
        notifyObservers();
    }

    /** @return int step from */
    public int getStepFrom() { return this.stepFrom; }

    /** @param int new step from */
    public void setStepFrom (int newStepFrom) {
        this.stepFrom = newStepFrom;
        // must call setChanged before notifyObservers to
        // indicate model has changed
        setChanged();

        // notify Observers that model has changed
        notifyObservers();
    }

    /** @return int step to */
    public int getStepTo() {
        return this.stepTo;
    }

    /** @param int new step from */
    public void setStepTo (int newStepTo) {
        this.stepTo = newStepTo;
        // must call setChanged before notifyObservers to
        // indicate model has changed
        setChanged();

        // notify Observers that model has changed
        notifyObservers();
    }

    /** @return Point step position */
    public Point getStepPosition(){ return this.stepPosition; }

    /** @param Point p - new step position */
    public void setStepPosition(Point p) { this.stepPosition = p; }

    /** @return Hashtable of step QFD information */
    public Hashtable getStepQFDHashtable(){ return this.stepQFDHashtable; }

    /** @param Hashtable sets the QFD hashtable information */
    public void setStepQFDHashtable(Hashtable stepQFDHashtable){ this.stepQFDHashtable = stepQFDHashtable; }
}

```

23. Cases.TraceFrame

```

/**-----
 * @Filename: TraceFrame.java
 * @Date: 3-7-2003
 * @Author: Le Hahn but modified by Arthur Clomera for NPS Thesis
 * Description: the main purpose of this frame is to view and trace the step and
 * substeps

```

```

* Implement CasesTitle where stores all global variables of Cases package
* Implements interface I_Trace
* Removed dependencies on Visual Cafe.
* Modified with standard event handlers and listeners.
* @Compiler: JDK 1.3.1
* -----
**/
package Cases;

import java.awt.*;
import java.awt.event.ActionListener;
import java.io.*;
import java.util.StringTokenizer;
import java.util.Vector;
import javax.swing.*;
import Cases.Interfaces.I_Trace;

////////////////////////////////////
/**
 * TraceFrame : the main purpose of this frame is to view and trace the step and
 * substeps
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_Trace
 */
////////////////////////////////////
public class TraceFrame extends javax.swing.JFrame implements CasesTitle, I_Trace, ActionListener
{
    /**
     * casesFrame : parent frame which launch this frame
     */
    CasesFrame casesFrame = null;

    /**
     * storedVector : stores contents of link files
     */
    public Vector[] storedVector = {new Vector(), new Vector(), new Vector(), new Vector(), new Vector(), new Vector(), new
    Vector()};

    /**
     * currentIndex : monitor the position of current step in the history vector
     */
    public int currentIndex = 0;

    /**
     * currentLocation : make sure the current step is inserted at the right position in the history vector
     */
    public int currentLocation = 0;

    /**
     * pathName : the path of the current project
     */
    public String pathName = null;

    /**
     * history : a vector of all selected steps in this frame
     */
    public Vector history = new Vector();

    /**
     * fnList : a vector of steps, strings, in the current project
     */
    public Vector fnList = new Vector();

    /**
     * currentPath : a string to keep the current step when tracing around
     */
    public String currentPath = null;

```

```

/**
 * output : holds the output of the current step
 */
private String output = "";

/**
 * stepVersion : holds the selected step version
 */
private String stepVersion = "";

/**
 * Build TraceFrame
 */
public TraceFrame() {
    initGUI();
}

/**
 * this method initializes the GUI and its components
 */
private void initGUI() {

    //{ INIT_CONTROLS
    setTitle("SPIDER-Trace");
    getContentPane().setLayout(null);
    setSize(742,320);
    setVisible(false);
    forwardButton.setText("Forward");
    forwardButton.setActionCommand("Forward");
    getContentPane().add(forwardButton);
    forwardButton.setBounds(68,1,81,24);
    backwardButton.setText("Backward");
    backwardButton.setActionCommand("Backward");
    getContentPane().add(backwardButton);
    backwardButton.setBounds(150,1,92,24);
    homeButton.setText("Home");
    homeButton.setActionCommand("Home");
    getContentPane().add(homeButton);
    homeButton.setBounds(1,1,67,24);
    stepVersionTextField.setEditable(false);
    getContentPane().add(stepVersionTextField);
    stepVersionTextField.setBackground(java.awt.Color.white);
    stepVersionTextField.setBounds(242,55,300,24);
    outputTextField.setEditable(false);
    getContentPane().add(outputTextField);
    outputTextField.setBackground(java.awt.Color.white);
    outputTextField.setBounds(242,100,300,24);
    JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    JLabel1.setText("Step Version");
    getContentPane().add(JLabel1);
    JLabel1.setForeground(java.awt.Color.black);
    JLabel1.setBounds(55,55,180,24);
    JLabel2.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    JLabel2.setText("Primary Input Component(s)");
    getContentPane().add(JLabel2);
    JLabel2.setForeground(java.awt.Color.black);
    JLabel2.setBounds(55,145,180,24);
    JLabel3.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    JLabel3.setText("Secondary Input Component(s)");
    getContentPane().add(JLabel3);
    JLabel3.setForeground(java.awt.Color.black);
    JLabel3.setBounds(55,190,180,24);
    JLabel4.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    JLabel4.setText("Output Component");
    getContentPane().add(JLabel4);
    JLabel4.setForeground(java.awt.Color.black);
    JLabel4.setBounds(55,100,180,24);
}

```

```

        closeButton.setText("Close");
        closeButton.setActionCommand("OK");
        getContentPane().add(closeButton);
        closeButton.setBounds(302,252,75,24);
        stepContentButton.setText("Step Content");
        stepContentButton.setActionCommand("Step Content");
        getContentPane().add(stepContentButton);
        stepContentButton.setBounds(632,1,107,24);
        primaryTextField.setEditable(false);
        getContentPane().add(primaryTextField);
        primaryTextField.setBackground(java.awt.Color.white);
        primaryTextField.setBounds(242,145,300,24);
        secondaryTextField.setEditable(false);
        getContentPane().add(secondaryTextField);
        secondaryTextField.setBackground(java.awt.Color.white);
        secondaryTextField.setBounds(242,190,300,24);
        traceButton.setText("Trace");
        traceButton.setActionCommand("Trace");
        getContentPane().add(traceButton);
        traceButton.setBounds(242,1,67,24);
        decomposeButton.setText("Decompose");
        decomposeButton.setActionCommand("Decompose");
        getContentPane().add(decomposeButton);
        decomposeButton.setBounds(382,1,102,24);
        componentButton.setText("Component Content");
        componentButton.setActionCommand("Component Content");
        getContentPane().add(componentButton);
        componentButton.setBounds(485,1,146,24);
        parentButton.setText("Parent");
        getContentPane().add(parentButton);
        parentButton.setBounds(310,1,72,24);
    //} }

    //{{INIT_MENUS
    //}}

    //{{REGISTER_LISTENERS

        homeButton.addActionListener(this);
        forwardButton.addActionListener(this);
        backwardButton.addActionListener(this);
        closeButton.addActionListener(this);
        stepContentButton.addActionListener(this);
        traceButton.addActionListener(this);
        decomposeButton.addActionListener(this);
        componentButton.addActionListener(this);
        parentButton.addActionListener(this);
    //} }

    //Grey out when the history vector is empty
    forwardButton.setEnabled( false );
    backwardButton.setEnabled( false );
}

/**
 * CasesFrame launch this frame when Trace menu item is selected
 */
public TraceFrame( CasesFrame casesFrame, String pathName, Vector componentsVector, Vector fnList ){
    this();
    this.casesFrame = casesFrame;
    this.pathName = pathName;
    this.fnList = fnList;
    setInitial(componentsVector);
}

/**
 * constructor to set the title to sTitle
 * @param String

```

```

*/
    public TraceFrame(String sTitle) {
        this();
        setTitle(sTitle);
    }

/**
 * a method to control the visibility of the frame based upon b
 * @param boolean
 */
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

/**
 * a main procedure for unit testing
 */
    static public void main(String[] args) {
        (new TraceFrame()).setVisible(true);
    }

/**
 * overrides the super addNotify()method
 */
    public void addNotify() {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    //{DECLARE_CONTROLS
    javax.swing.JButton forwardButton = new javax.swing.JButton();
    javax.swing.JButton backwardButton = new javax.swing.JButton();
    javax.swing.JButton homeButton = new javax.swing.JButton();
    javax.swing.JTextField stepVersionTextField = new javax.swing.JTextField();
    javax.swing.JTextField outputTextField = new javax.swing.JTextField();
    javax.swing.JLabel JLabel1 = new javax.swing.JLabel();
    javax.swing.JLabel JLabel2 = new javax.swing.JLabel();
    javax.swing.JLabel JLabel3 = new javax.swing.JLabel();
    javax.swing.JLabel JLabel4 = new javax.swing.JLabel();
    javax.swing.JButton closeButton = new javax.swing.JButton();
    javax.swing.JButton stepContentButton = new javax.swing.JButton();
    javax.swing.JTextField primaryTextField = new javax.swing.JTextField();
    javax.swing.JTextField secondaryTextField = new javax.swing.JTextField();
    javax.swing.JButton traceButton = new javax.swing.JButton();
    javax.swing.JButton decomposeButton = new javax.swing.JButton();
    javax.swing.JButton componentButton = new javax.swing.JButton();
    javax.swing.JButton parentButton = new javax.swing.JButton();
    //}

```



```

        //{{DECLARE_MENUS
        //}}

/**
 * handles standard action events
 */
    public void actionPerformed(java.awt.event.ActionEvent event)    {
        Object object = event.getSource();
        if (object == homeButton) // home button pressed
            homeButton_actionPerformed(event);
        else if (object == closeButton) // close button pressed
            closeButton_actionPerformed(event);
        else if (object == forwardButton) // forward button pressed
            forwardButton_actionPerformed(event);
        else if (object == backButton) // backward button pressed
            backButton_actionPerformed(event);
        else if (object == stepContentButton) // step content button pressed
            stepContentButton_actionPerformed(event);
        else if (object == traceButton) // trace button pressed
            traceButton_actionPerformed(event);
        else if (object == decomposeButton) // decompose button pressed
            decomposeButton_actionPerformed(event);
        else if (object == componentButton) // component button pressed
            componentButton_actionPerformed(event);
        else if (object == parentButton) // parent button pressed
            parentButton_actionPerformed(event);
    }

/**
 * View the first step in the history vector
 */
    public void homeButton_actionPerformed(java.awt.event.ActionEvent event)
    {
        this.currentIndex = 0;
        String s = (String)this.history.elementAt(this.currentIndex);
        this.currentPath = (String)convertToThePath(s);
        System.out.println("TraceFrame:homeButton:currentPath:"+currentPath);
        searchPath(s);
    }

/**
 * View the step at the currentIndex+1 postion in the history vector
 */
    public void forwardButton_actionPerformed(java.awt.event.ActionEvent event)
    {
        if( (++this.currentIndex >= 0 ) && (this.currentIndex < this.history.size()) ){
            String s = (String)this.history.elementAt(this.currentIndex);
            this.currentPath = (String)convertToThePath(s);
            System.out.println("TraceFrame:forwardButton:currentPath:"+currentPath);
            searchPath(s);
        }
    }

/**
 * View the step at the currentIndex-1 postion in the history vector
 */
    public void backButton_actionPerformed(java.awt.event.ActionEvent event)
    {
        if ( --this.currentIndex >= 0 ){
            String s = (String)this.history.elementAt(this.currentIndex);
            this.currentPath = (String)convertToThePath(s);
            System.out.println("TraceFrame:backwardButton:currentPath:"+currentPath);
            searchPath(s);
        }
    }

/**

```

```

* Launch StepContentFrame to view the content of this current step
*/
    public void stepContentButton_actionPerformed(java.awt.event.ActionEvent event)
    {
        String s = (String)this.history.elementAt(this.currentIndex);
        this.currentPath = (String)convertToThePath(s);
        System.out.println("TraceFrame:stepContentButton:currentPath:"+currentPath);
        this.casesFrame.setStepContent(this, "s-"+s, this.currentPath );
    }

/**
 * Exit TraceFrame
 */
    public void closeButton_actionPerformed(java.awt.event.ActionEvent event)
    {
        setVisible(false);
        dispose();
    }

/**
 * Short cut to print the output
 * @param string : the output string
 */
    public void debug(String s){
        System.out.println(s);
    }

/**
 * Set initial of this frame with the selected step from JFileChooser
 *
 * @param componentsVector : a vector of strings holds all component names of the selected step
 */
public void setInitial(Vector componentsVector){
    String s = (String)componentsVector.elementAt(0);
    setHistory(s);
    this.currentPath = (String)convertToThePath(s);
    System.out.println("TraceFrame:setInitial:currentPath:"+currentPath);
    if( this.currentPath != null ){
        searchPath(s);
    }
}

/**
 * Hold all steps which have been viewed and traced
 *
 * @param s : the current step name
 */
public void setHistory( String s){
    this.history.insertElementAt(s,currentLocation++);
}

/**
 * Convert from a selected step into the complete file path of this step and
 * make sure this step is valid or not
 *
 * @param s : selected step name
 */
public String convertToThePath(String s){
    String thePath = null;
    output = s;
    s= "s-"+s;
    stepVersion = s;
    System.out.println("TraceFrame:convertToThePath:stepVersion: " +stepVersion);

    //To convert the string of selected item into the file path
    for( int i=0; i<this.fnList.size(); i++ ){
        String fn = (String)this.fnList.elementAt(i);
        String sub = s.substring(0,fn.length());

```

```

        if( sub.equals(fn)){
            i = this.fnList.size();
            thePath= this.pathName+"\\ "+sub+"\\ ";
            String sub2 = s.substring(fn.length());

            char[] ca = (char[])sub2.toCharArray();
            int result = 0;
            for( int j=0; j<ca.length; j++ ){
                String s3 = ca[j]+" ";
                result = s3.compareTo("- ");
                if( result == 0 ){
                    j=ca.length;
                    StringTokenizer st = new StringTokenizer(sub2,"- ");
                    String before_ = st.nextToken("- ");
                    thePath = thePath+before_;
                    String _after = st.nextToken("- ");
                    st = new StringTokenizer(_after," ");
                    while(st.hasMoreTokens()){
                        thePath = thePath+"\\ "+st.nextToken();
                    }
                }
                else if( (result > 0) && (j==ca.length - 1) ){
                    thePath = thePath+sub2;
                }
            }
        }
    }
    if( thePath == null ){
        JOptionPane.showMessageDialog(this, s+" is invalid name.",
            "Error Message",JOptionPane.ERROR_MESSAGE);
        return null;
    }
    return thePath;
}

/**
 * Search the path of this step
 *
 * @param s : the selected step name
 */
public void searchPath( String s){
    System.out.println("TraceFrame:searchPath...");
    searchIndex();
    if( this.currentPath != null ){
        searchInputFiles( this.currentPath );
    }
}

/**
 * Search input.p and input.s files of the current step, and get theirs
 * contents to insert in the primary and secondary textfields directly
 *
 * @param thePath : the complete path of the selected step
 */
public void searchInputFiles( String thePath ){
    System.out.println("TraceFrame:searchInputFiles...");
    clearTextFields();
    try{
        System.out.println("TraceFrame:searchInputFiles:thePath:"+thePath);
        File f = new File(thePath+"\\input.p");
        if( f.exists() ){
            FileInputStream fileInputP = new FileInputStream(f);
            DataInputStream primIn = new DataInputStream( fileInputP );
            if( primIn != null ){
                String s = (String)primIn.readLine();
                System.out.println("TraceFrame:searchInputFiles:s:"+s);
                if( (s != null) && (!s.equals("")) ){

```

```

        this.primaryTextField.setText( s );
    }
}
primIn.close();
fileInputP.close();
}
f = new File(thePath+"\\input.s");
if( f.exists() ){
    FileInputStream fileInputS = new FileInputStream(f);
    DataInputStream secondIn = new DataInputStream( fileInputS);
    if( secondIn != null ){
        String s = (String)secondIn.readLine();
        if( s != null ) && (!s.equals("")) ){
            this.secondaryTextField.setText( s );
        }
    }
    secondIn.close();
    fileInputS.close();
}
this.outputTextField.setText(output);
this.stepVersionTextField.setText(stepVersion);
if( stepVersion.length() > 13 ){ // an atomic step name is 13 characters long
    parentButton.setEnabled( true);
}
else{
    parentButton.setEnabled(false);
}
        }
        catch( IOException io ){
            debug("IOException: "+io);
        }
    }
}

/**
 * To find which button (home, forward, or backward) should be set enabled
 * and it must match with the currentIndex in history vector
 */
public void searchIndex(){
    System.out.println("TraceFrame:searchInputIndex...");
    if( this.currentIndex < this.history.size() ){
        if( this.currentIndex == (this.history.size()-1) ){
            this.forwardButton.setEnabled( false );
        }
        else{
            this.forwardButton.setEnabled( true );
        }
    }
    if( this.currentIndex > 0 ){
        this.backwardButton.setEnabled( true );
    }
    else{
        this.backwardButton.setEnabled( false );
    }
}

/**
 * Get selected item from ListDialog and DecomposeListDialog
 *
 * @param currentPath : reverse from a selected item into the path of this component
 * @param selectedItem : a selected component names
 */
public void setSelectedItem(String currentPath, String selectedItem){
    System.out.println("TraceFrame:setSelectedItem...");

    this.currentPath = currentPath;
    System.out.println("TraceFrame:setSelectedItem:currentPath:"+currentPath);
    setHistory(selectedItem);
    this.currentIndex = this.history.size() - 1;
}

```

```

        searchPath(currentPath);
    }

    /**
     * Searching aFile and get its content to insert into one of elements of storedVector
     *
     * @param aFile : link files
     * @param fileType : type of link file, eg. txt.link, word.link, ...
     */
    public void searchFiles(File aFile, String fileType){
        System.out.println("TraceFrame:searchFiles...");
        File f = new File( aFile, fileType);
        try{
            BufferedReader br = null;
            String item = null;
            if( fileType.equals(LINK_FILE_NAMES[0]) ){
                br = new BufferedReader( new FileReader(f));
                if( br != null ){
                    while( (item = br.readLine()) != null ){
                        this.storedVector[0].addElement(item);
                    }
                }
            }
            else if( fileType.equals(LINK_FILE_NAMES[1]) ){
                br = new BufferedReader( new FileReader(f));
                if( br != null ){
                    while( (item = br.readLine()) != null ){
                        this.storedVector[1].addElement(item);
                    }
                }
            }
            else if( fileType.equals(LINK_FILE_NAMES[2]) ){
                br = new BufferedReader( new FileReader(f));
                if( br != null ){
                    while( (item = br.readLine()) != null ){
                        this.storedVector[2].addElement(item);
                    }
                }
            }
            else if( fileType.equals(LINK_FILE_NAMES[3]) ){
                br = new BufferedReader( new FileReader(f));
                if( br != null ){
                    while( (item = br.readLine()) != null ){
                        this.storedVector[3].addElement(item);
                    }
                }
            }
            else if( fileType.equals(LINK_FILE_NAMES[4]) ){
                br = new BufferedReader( new FileReader(f));
                if( br != null ){
                    while( (item = br.readLine()) != null ){
                        this.storedVector[4].addElement(item);
                    }
                }
            }
            else if( fileType.equals(LINK_FILE_NAMES[5]) ){
                br = new BufferedReader( new FileReader(f));
                if( br != null ){
                    while( (item = br.readLine()) != null ){
                        this.storedVector[5].addElement(item);
                    }
                }
            }
            else if( fileType.equals(LINK_FILE_NAMES[6]) ){
                br = new BufferedReader( new FileReader(f));
                if( br != null ){
                    while( (item = br.readLine()) != null ){
                        this.storedVector[6].addElement(item);
                    }
                }
            }
        }
    }

```

```

        }
    }
}
catch( IOException io ){
    debug("IOException: "+io);
}
}

/**
 * Create a file with a specific string s if s is valid
 *
 * @param s : a selected string
 * @return f : a file with the name is s
 */
public File searchFilePath( String s){
    System.out.println("TraceFrame:searchFilePath...");
    String thePath = null;

    s = "s-"+s;
    //To convert the string of selected item into the file path
    for( int i=0; i<this.fnList.size(); i++ ){
        String fn = (String)this.fnList.elementAt(i);
        String sub = s.substring(0,fn.length());

        if( sub.equals(fn)){
            i = this.fnList.size();
            thePath= sub+"\\";
            String sub2 = s.substring(fn.length());

            char[] ca = (char[])sub2.toCharArray();
            int result = 0;
            for( int j=0; j<ca.length; j++ ){
                String s3 = ca[j]+"";
                result = s3.compareTo("-");
                if( result == 0 ){
                    j=ca.length;
                    StringTokenizer st = new StringTokenizer(sub2,"-");
                    String before_ = st.nextToken("-");
                    thePath = thePath+before_ ;
                    String _after = st.nextToken("-");
                    st = new StringTokenizer(_after,".");
                    while(st.hasMoreTokens()){
                        thePath = thePath+"\\st.nextToken();
                    }
                }
            }
            else if( (result > 0) && (j==ca.length - 1) ){
                thePath = thePath+sub2;
            }
        }
    }
    File f = new File(this.pathName,thePath);
    return f;
}

/**
 * Check a selected string is valid or invalid file name in the current project
 *
 * @param selectedItem : a selected component name
 * @return a component name which matchs in the current project
 */
public String checkSelection(String selectedItem ){
    System.out.println("TraceFrame:checkSelection...");
    if( selectedItem != null ){
        int j = selectedItem.indexOf("-");
        if( j>0){
            String sub2 = (selectedItem.substring(j+1)).trim();

```

```

        char c = (char)sub2.charAt(0);
        boolean isLetter = (new Character(c)).isLetter(c);
        if( isLetter ){
            return sub2;
        }
        else{
            return selectedItem;
        }
    }
    else{
        return selectedItem;
    }
}
return selectedItem;
}

/**
 * Tokenize a string with the delimit is ","
 *
 * @param v : a vector of words without ","
 * @param s : a string is tokenized
 */
public void tokenizer( Vector v, String s){
    System.out.println("TraceFrame:tokenizer...");
    StringTokenizer st = new StringTokenizer(s, ",");
    while(st.hasMoreTokens()){
        String theString = (st.nextToken()).trim();
        v.addElement(theString);
    }
}

/**
 * View the content of available steps in the ListDialog
 */
public void traceButton_actionPerformed(java.awt.event.ActionEvent event)
{
    Vector v = new Vector();
    System.out.println("TraceFrame:traceButton:primaryTextField:"+primaryTextField.getText());
    tokenizer(v, primaryTextField.getText());
    System.out.println("TraceFrame:traceButton:secondaryTextField:"+secondaryTextField.getText());
    tokenizer(v, secondaryTextField.getText());
    (new ListDialog(this, "Trace Component", v, 0)).setVisible(true);
}

/**
 * View the content of available decomposed steps of the current step
 */
public void decomposeButton_actionPerformed(java.awt.event.ActionEvent event)
{
    Vector v = new Vector();
    v.addElement(outputTextField.getText());
    tokenizer(v, primaryTextField.getText());
    tokenizer(v, secondaryTextField.getText());
    (new DecomposeListDialog(this, "Decompose", v)).setVisible(true);
}

/**
 * View all links and connect these links to their applications
 * of available steps in ListDialog
 */
public void componentButton_actionPerformed(java.awt.event.ActionEvent event)
{
    Vector v = new Vector();
    v.addElement(outputTextField.getText());
    tokenizer(v, primaryTextField.getText());
    tokenizer(v, secondaryTextField.getText());
    (new ListDialog(this, "Component Content", v, 1)).setVisible(true);
}

```

```

/**
 * Refresh all textfiels in this frame
 */
    public void clearTextFields(){
System.out.println("TraceFrame:clearTextFields...");
        this.stepVersionTextField.setText("");
        this.outputTextField.setText("");
        this.primaryTextField.setText("");
        this.secondaryTextField.setText("");
    }

/**
 * Launch ReviewComponentContentDialog after select one component from ListDialog
 *
 * @param selectedItem : a selected component name from ListDialog
 * @param f : file with the name is selectedItem
 */
    public void setComponentContent(String selectedItem, File f){
System.out.println("TraceFrame:setComponentContent...");
String[] list = f.list();
for( int j=0; j<list.length; j++ ){
    String s = (String)list[j];
    File aFile = null;
    if( s.equals(COMPONENT_CONTENT_DIR) ){
        aFile = new File(f, s);
        if( aFile.isDirectory() ){
            j = list.length;
            list = aFile.list();
            for( int k=0; k<storedVector.length; k++ ){
                storedVector[k] = new Vector();
            }
            for( int i=0; i<list.length; i++ ){
                searchFiles(aFile, (String)list[i]);
            }
            (new ReviewComponentContentDialog(selectedItem, this.storedVector)).setVisible(true);
        }
    }
}
}

void parentButton_actionPerformed(java.awt.event.ActionEvent event)
{
    String currentString = this.stepVersionTextField.getText();
    String newString = "";
    System.out.println("TraceFrame:parentButton:currentString.length:"+currentString.length());
System.out.println("TraceFrame:parentButton:currentString.lastIndexOf:"+currentString.lastIndexOf("-"));
    if( currentString.length() > 13 ){ // must not be atomic
        if( currentString.lastIndexOf("-") == 1 ){ //
            int lastDot = (int)currentString.lastIndexOf(".");
System.out.println("TraceFrame:parentButton:lastDot:"+lastDot);
            if( lastDot == 14 ){ // is this the only step with pattern s-abComponent1.1
System.out.println("TraceFrame:currentString: " +currentString);
                // strip off variant.version and s-, leaving the atomic
                newString = (String)currentString.substring(2,lastDot-1);
System.out.println("TraceFrame:newString: " +newString);
            }
            else if( lastDot > 14 ){ // must be large variant number (eg s-abComponent12.5)
System.out.println("TraceFrame:currentString: " +currentString);
                newString = (String)currentString.substring(2,lastDot);
System.out.println("TraceFrame:newString: " +newString);
            }
        }
    }
System.out.println("TraceFrame:parent_Button:newString:"+newString);
this.currentPath = (String)convertToThePath(newString);
System.out.println("TraceFrame:parentButton:currentPath:"+currentPath);
setHistory(newString);
searchInputFiles(this.currentPath);
}

```



```

    }
}
}

```

24. Cases.VersionControl

```

/**-----
 * @Filename: VersionControl.java
 * @Date: 3-7-2003
 * @Author: Le Hahn
 * Description: Create a version control object and save it in current.vsn file
 * @Compiler: JDK 1.3.1
 * -----
 */
package Cases;

/**
 * Version Control Object which is used to save in the
 * current.vsn file
 */

import java.io.Serializable;

////////////////////////////////////
/**
 * VersionControl : Create a version control object and save it in current.vsn file
 */
////////////////////////////////////
public class VersionControl implements Serializable{
    /**
     * currentLoop : current process of the project
     */
    private String currentLoop = null;

    /**
     * currentStep : current step of the current process
     */
    private String currentStep = null;

    /**
     * current variant : current variant of the current step
     */
    private String currentVariant = null;

    /**
     * currentVersion : current version of the current step
     */
    private String currentVersion = null;

    /**
     * currentStatus : current status of the current step, eg, Completed, Approved, ...
     */
    private String currentStatus = "";

    /**
     * VersionControl constructor with 5 elements
     */
    public VersionControl( String currentLoop, String currentStep, String currentVariant,
        String currentVersion, String currentStatus )
    {
        this.currentLoop = currentLoop;
        this.currentStep = currentStep;
        this.currentVariant = currentVariant;
        this.currentVersion = currentVersion;
    }
}

```

```

        this.currentStatus = currentStatus;
    }

    /**
     * VersionControl constructor with 4 elements
     */
    public VersionControl( String currentLoop, String currentStep, String currentVersion,
        String currentStatus )
    {
        this.currentLoop = currentLoop;
        this.currentStep = currentStep;
        this.currentVersion = currentVersion;
        this.currentStatus = currentStatus;
    }

    /**
     * VersionControl constructor with 3 elements
     */
    public VersionControl( String currentLoop, String currentStep, String currentVersion )
    {
        this.currentLoop = currentLoop;
        this.currentStep = currentStep;
        this.currentVersion = currentVersion;
    }

    public VersionControl() {}

    /**
     * @return currentLoop : current process of this version control object
     */
    public String getCurrentLoop(){

        return this.currentLoop;
    }

    /**
     * @return currentStep : current step of this version control object
     */
    public String getCurrentStep(){

        return this.currentStep;
    }

    /**
     * @return currentVersion : current version of this version control object
     */
    public String getCurrentVersion(){

        return this.currentVersion;
    }

    /**
     * @return currentVariant : current variant of this version control object
     */
    public String getCurrentVariant(){

        return this.currentVariant;
    }

    /**
     * @return currentStatus : current status of this version control object
     */
    public String getCurrentStatus(){

        return this.currentStatus;
    }
}

```

B. CASES.GUI PACKAGE

1. Cases.GUI.CasesToolBar

```
/**-----
 * Filename: CasesToolBar.java
 * @Date: 10-29-2002
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.
 * Description: the cases tool bar which provides functionality to graph pane
 *-----
 */

package Cases.GUI;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

import Cases.CasesFrame;
import Cases.CasesTitle;
import Cases.DependencyType;
import Cases.GUI.avc.AVCOpenStepFrame;
import Cases.QFD.UpdateOrigin;
import Cases.ProjectSchemaFrame;
import Cases.GUI.util.NextVersion;
import Cases.QFD.MyDefaultTableModel;
import Cases.QFD.ComponentQFD;
import Cases.QFD.StepQFD;
import Cases.ComponentType;
import Cases.StepType;
import java.util.Hashtable;
import Cases.QFD.HouseMatrix;
import Cases.GUI.util.PreviousVersion;

/**
 * this class extends the JToolBar and implement the CasesTitle with the global variables
 * and Action listener for mouse events.
 */
public class CasesToolBar extends JToolBar implements CasesTitle, ActionListener {
    /** a button to create dependencies */
    private JButton dependencyButton= new JButton();
    /** a button to select items on the draw pane */
    private JButton selectButton = new JButton();
    /** a button to draw steps */
    private JButton stepButton = new JButton();
    /** a button to delete items on the draw pane */
    private JButton deleteButton = new JButton();
    /** a button to draw components on the draw pane */
    private JButton componentButton = new JButton();
    /** a synchronize button to deploy dependency values */
    private JButton calcButton = new JButton();
    /** a variable to link to CasesFrame */
    protected CasesFrame parentFrame;
    /** a button to lock the project schema */
    private JButton lockButton = new JButton();
    /** a text field to display project name */
    private JTextField projectJTextField = new JTextField();

    /**
     * constructor
     * @param CasesFrame
     */
}
```

```

public CasesToolBar(CasesFrame frame) {
    parentFrame=frame;
    initGUI();
}
/**
 * procedure to initialize GUI and its components
 */
public void initGUI() {
    // create drawing buttons
    componentButton.setActionCommand("componentButton");
    componentButton.setIcon(new javax.swing.ImageIcon("/CASES/IMAGES/COMPONENT.GIF"));
    componentButton.setLabel("Component");
    componentButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    componentButton.setToolTipText("Component");
    componentButton.setFont(new java.awt.Font("SansSerif", java.awt.Font.PLAIN, 8));
    componentButton.setBounds(new java.awt.Rectangle(16,2,467,50));

    stepButton.setActionCommand("stepButton");
    stepButton.setIcon(new javax.swing.ImageIcon("/CASES/IMAGES/STEP.GIF"));
    stepButton.setLabel("Step");
    stepButton.setToolTipText("Step");
    stepButton.setRolloverEnabled(true);
    stepButton.setRolloverIcon(new javax.swing.ImageIcon("/CASES/IMAGES/STEP.GIF"));
    stepButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    stepButton.setBounds(new java.awt.Rectangle(63,50,375,204));

    dependencyButton.setLabel("");
    dependencyButton.setRolloverEnabled(true);
    dependencyButton.setRolloverIcon(new javax.swing.ImageIcon("/CASES/IMAGES/Dep.GIF"));
    dependencyButton.setIcon(new javax.swing.ImageIcon("/CASES/IMAGES/Dep.GIF"));
    dependencyButton.setActionCommand("DependencyButton");
    dependencyButton.setToolTipText("Dependency");
    dependencyButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    dependencyButton.setBounds(new java.awt.Rectangle(16,251,467,50));

    selectButton.setActionCommand("selectButton");
    selectButton.setIcon(new javax.swing.ImageIcon("/CASES/IMAGES/SELECT.GIF"));
    selectButton.setLabel("Select");
    selectButton.setToolTipText("Select");
    selectButton.setRolloverEnabled(true);
    selectButton.setRolloverIcon(new javax.swing.ImageIcon("/CASES/IMAGES/SELECT.GIF"));
    selectButton.setMaximumSize(new java.awt.Dimension(50, 50));
    selectButton.setMinimumSize(new java.awt.Dimension(50, 50));
    selectButton.setPreferredSize(new java.awt.Dimension(50, 50));
    selectButton.setVerticalAlignment(javax.swing.SwingConstants.TOP);
    selectButton.setFont(new java.awt.Font("SansSerif", java.awt.Font.BOLD, 10));
    selectButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    selectButton.setVerticalTextPosition(javax.swing.SwingConstants.CENTER);
    selectButton.setBorderPainted(true);
    selectButton.setBounds(new java.awt.Rectangle(438,50,50,204));

    deleteButton.setActionCommand("deleteButton");
    deleteButton.setIcon(new javax.swing.ImageIcon("/CASES/IMAGES/delete.GIF"));
    deleteButton.setLabel("Delete");
    deleteButton.setRolloverIcon(new javax.swing.ImageIcon("/CASES/IMAGES/delete.GIF"));
    deleteButton.setToolTipText("Delete");
    deleteButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    deleteButton.setBounds(new java.awt.Rectangle(16,50,50,204));
    setLayout(new javax.swing.BoxLayout(this, javax.swing.BoxLayout.X_AXIS));

    calcButton.setActionCommand("calcButton");
    calcButton.setIcon(new javax.swing.ImageIcon("/CASES/IMAGES/synchronize.GIF"));
    calcButton.setLabel("Synch");
    calcButton.setRolloverIcon(new javax.swing.ImageIcon("/CASES/IMAGES/synchronize.GIF"));
    calcButton.setToolTipText("Update all calculations");
    calcButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    calcButton.setBounds(new java.awt.Rectangle(16,50,50,204));
    calcButton.setMaximumSize(new java.awt.Dimension(45, 45));

```

```

        calcButton.setMinimumSize(new java.awt.Dimension(45, 45));
        calcButton.setContentAreaFilled(true);
        calcButton.setDefaultCapable(true);
        setBounds(new java.awt.Rectangle(0, 0, 650, 67));

add(selectButton);
add(componentButton);
add(stepButton);
add(dependencyButton);
add(deleteButton);
add(calcButton);
add(lockButton);
add(projectJTextField);
selectButton.addActionListener(this);
selectButton.setEnabled(false);
componentButton.addActionListener(this);
componentButton.setEnabled(false);
stepButton.addActionListener(this);
stepButton.setEnabled(false);
deleteButton.addActionListener(this);
deleteButton.setEnabled(false);
calcButton.addActionListener(this);
calcButton.setEnabled(false);
dependencyButton.addActionListener(this);
dependencyButton.setEnabled(false);
lockButton.addActionListener(this);

lockButton.setText("jButton1");
lockButton.setMaximumSize(new java.awt.Dimension(45,45));
lockButton.setMinimumSize(new java.awt.Dimension(45,45));
lockButton.setActionCommand("lockButton");
lockButton.setLabel("Lock");
lockButton.setEnabled(false);
lockButton.setPressedIcon(new javax.swing.ImageIcon("C:/CASES/IMAGES/LOCK.gif"));
lockButton.setIcon(new javax.swing.ImageIcon("C:/CASES/IMAGES/LOCK.gif"));
lockButton.setRolloverSelectedIcon(new javax.swing.ImageIcon("C:/CASES/IMAGES/LOCK.gif"));
lockButton.setSelectedIcon(new javax.swing.ImageIcon("C:/CASES/IMAGES/LOCK.gif"));
lockButton.setRolloverIcon(new javax.swing.ImageIcon("C:/CASES/IMAGES/LOCK.gif"));
lockButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
lockButton.setFont(new java.awt.Font("SansSerif", java.awt.Font.BOLD, 12));
projectJTextField.setText("");
projectJTextField.setBackground(new java.awt.Color(212, 208, 200));
projectJTextField.setEditable(false);
projectJTextField.setFont(new java.awt.Font("SansSerif", java.awt.Font.BOLD, 14));
projectJTextField.setHorizontalAlignment(javax.swing.JTextField.LEFT);
projectJTextField.setMaximumSize(new java.awt.Dimension(200, 45));
projectJTextField.setMinimumSize(new java.awt.Dimension(200, 45));
projectJTextField.setPreferredSize(new java.awt.Dimension(150, 45));
projectJTextField.setSize(new java.awt.Dimension(45, 150));
projectJTextField.setToolTipText("Project Name");
projectJTextField.setVerifyInputWhenFocusTarget(false);
projectJTextField.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.border.BevelBorder.LOWERED));
}

/**
 * returns the Select button
 * @return JButton
 */
public JButton getSelectButton() { return selectButton; }

/**
 * returns the step button
 * @return JButton
 */
public JButton getStepButton() { return stepButton; }

/**
 * returns the component button

```

```

    * @return JButton
    */
    public JButton getComponentButton() { return componentButton;}

    /**
    * returns the delete button
    * @return JButton
    */
    public JButton getDeleteButton() { return deleteButton; }

    /**
    * returns the dependency button
    * @return JButton
    */
    public JButton getDependencyButton() { return dependencyButton; }

    /**
    * a method to set the project name in the text field
    * @param String
    */
    public void setProjectName (String projectName) {
        this.projectJTextField.setText(projectName);
    }

    /** a procedure which enables or disables the a set of buttons based on flag
    * @param boolean
    */
    public void setButtons (boolean flag) {
        componentButton.setEnabled(flag);
        stepButton.setEnabled(flag);
        deleteButton.setEnabled(flag);
        dependencyButton.setEnabled(flag);
        selectButton.setEnabled(flag);
        calcButton.setEnabled(flag);
        lockButton.setEnabled(flag);
    }

    /**
    * enables or disables the calculate button based on flag
    * @param boolean
    */
    public void setCalcButton (boolean flag) { calcButton.setEnabled(flag); }

    /**
    * complete the downstream calculation on a specified version
    * and dependency based upon the origin and dependency version
    * Gets information form avcFrame.
    * @param AVCOpenFrame, String, and int
    */
    private void downstreamCalcButtonActionPerformed (AVCOpenStepFrame avcFrame, String version, int dep) {
        // base case -- stop if there isn't a downstream version
        String newStep = parentFrame.drawPanel.stepID[parentFrame.drawPanel.numComponents-1][0];
        NextVersion nv = new NextVersion (avcFrame, newStep, version );
        if (nv.getNextVersion() != "END") { // "END" is used to mark the last version
            // table models to construct a HOQ
            MyDefaultTableModel leftTable;
            MyDefaultTableModel topTable;
            MyDefaultTableModel matrix;
            String name = ((DependencyType)parentFrame.drawPanel.depAttrib.get(dep)).getName();
            String lastComponent = parentFrame.drawPanel.componentID[parentFrame.drawPanel.numComponents-1];
            String firstComponent ="aComponent";
            // get table data for matrices
            leftTable = ((ComponentQFD)((ComponentType)parentFrame.psfWindow.compHashtable.get(lastComponent
            )).getComponentHashtable().get(version)).getComponentTable(dep);
            topTable =
            ((ComponentQFD)((ComponentType)parentFrame.psfWindow.compHashtable.get(firstComponent
            )).getComponentHashtable().get(nv.getNextVersion()))).getComponentTable(dep);

```

```

        matrix =
((StepQFD)((Hashtable)((StepType)parentFrame.psfWindow.stepHashtable.get(newStep)).getStepQFDHashtable()).get(version)).get
StepTable();
        // construct the hoq
        HouseMatrix downHouse = new HouseMatrix(parentFrame.projectName, name+" v."+version+"-
>" + nv.getNextVersion(),
                                                    dep,
parentFrame.drawPanel.numComponents, lastComponent, leftTable, topTable, matrix,
        lastComponent, firstComponent, nv.getNextVersion());
        downHouse.calcButtonActionPerformed(); // update calculations on the matrix
        downHouse.saveButtonActionPerformed(); // save the updates
        // update the rest of the matrices
        UpdateOrigin ufo = new UpdateOrigin (parentFrame.psfWindow, nv.getNextVersion(),
parentFrame.drawPanel.numComponents, firstComponent, dep);
        // update upstream matrices
        ufo.updateUpStreamMatrix(firstComponent);
        // update downstream matrices
        ufo.updateDownStreamMatrix(firstComponent);
        // no need for HOQ anymore
        downHouse.dispose();
        // recursive call to perform downstream calculations on the next version
        downstreamCalcButtonActionPerformed(avcFrame, nv.getNextVersion(), dep);
    }
}
/**
 * complete the upstream calculation on a specified version and dependency based upon the origin and dependency version
 * gets information from avcFrame
 * @param AVCOpenStepFrame, String, and int
 */
    private void upstreamCalcButtonActionPerformed (AVCOpenStepFrame avcFrame, String version, int dep) {
        // base case -- stop if there isn't a upstream version
        String newStep = parentFrame.drawPanel.stepID[parentFrame.drawPanel.numComponents-1][0];
        PreviousVersion nv = new PreviousVersion (avcFrame, newStep, version );
        if (nv.getPreviousVersion() != "END") { // "END" is a marker that there are no more previous versions
            // table models for constructing HOQ
            MyDefaultTableModel leftTable;
            MyDefaultTableModel topTable;
            MyDefaultTableModel matrix;
            String name = ((DependencyType)parentFrame.drawPanel.depAttrib.get(dep)).getName();
            String firstComponent = "aComponent"; // default start node
            String lastComponent
=parentFrame.drawPanel.componentID[parentFrame.drawPanel.numComponents-1];
            // get table data
            leftTable = ((ComponentQFD)((ComponentType)parentFrame.psfWindow.compHashtable.get(lastComponent
)).getComponentHashtable().get(version)).getComponentTable(dep);
            topTable =
((ComponentQFD)((ComponentType)parentFrame.psfWindow.compHashtable.get(firstComponent
)).getComponentHashtable().get(nv.getPreviousVersion())).getComponentTable(dep);
            matrix =
((StepQFD)((Hashtable)((StepType)parentFrame.psfWindow.stepHashtable.get(newStep)).getStepQFDHashtable()).get(version)).get
StepTable();
            // construct HOQ
            HouseMatrix upHouse = new HouseMatrix(parentFrame.projectName, name+" v."+version+"-
>" + nv.getPreviousVersion(),
                                                    dep,
parentFrame.drawPanel.numComponents, firstComponent, leftTable, topTable, matrix,
        firstComponent, lastComponent, nv.getPreviousVersion());
            upHouse.calcButtonActionPerformed(); // update current matrix
            upHouse.saveButtonActionPerformed(); // save updates
            // deploy dependencies to all matrices
            UpdateOrigin ufo = new UpdateOrigin (parentFrame.psfWindow, nv.getPreviousVersion(),
parentFrame.drawPanel.numComponents, lastComponent, dep);
            // perform all upstream calculations
            ufo.updateUpStreamMatrix(lastComponent);
            // perform all downstream calculations
            ufo.updateDownStreamMatrix(lastComponent);
            // HOQ is no longer needed.
            upHouse.dispose();

```

```

        // recursive call to upstream calculations for previous version
        upstreamCalcButtonActionPerformed(avcFrame, nv.getPreviousVersion(), dep);
    }
}

/**
 * procedure for tracking action events performed
 */
public void actionPerformed (ActionEvent e) {
    if (e.getSource()==deleteButton) {
        parentFrame.drawPanel.gpf.setDeleteFlag();
    }
    if (e.getSource()==dependencyButton) {
        // open the dependency attribute dialog window
        parentFrame.openDependencyFrame();
    }
    if (e.getSource()==selectButton) {
        parentFrame.drawPanel.gpf.setSelectFlag();
    }
    if (e.getSource()==stepButton) {
        parentFrame.drawPanel.gpf.setStepFlag();
    }
    if (e.getSource()==componentButton) {
        parentFrame.drawPanel.gpf.setComponentFlag();
    }
    if (e.getSource()==lockButton) {
        /** automatically update EHL */
        parentFrame.psfWindow.updateButton_actionPerformed(null);
        /** automatically update primary and secondary dependencies */
        parentFrame.psfWindow.depenUpdateButton_actionPerformed(null);
    }

    if (e.getSource()==calcButton) {
        ProjectSchemaFrame psf = parentFrame.psfWindow;
        int lastComponent = parentFrame.drawPanel.numComponents-1;
        UpdateOrigin ufo;
        AVCOpenStepFrame avcFrame = new AVCOpenStepFrame(parentFrame.projectName);

        // perform calculations at the origin's version
        for (int i=0; i<parentFrame.drawPanel.depAttrib.size(); i++) {
            String depOrigin =
                ((DependencyType)parentFrame.drawPanel.depAttrib.get(i)).getOrigin();
            String versionNumber =((DependencyType)
                parentFrame.drawPanel.depAttrib.get(i)).getDepVersion();
            // System.out.println("CasesToolBar: "+((DependencyType)parentFrame.drawPanel.depAttrib.get(i)).getName() + " " +
            versionNumber);

            ufo = new UpdateOrigin (parentFrame.psfWindow, versionNumber,
                parentFrame.drawPanel.numComponents,depOrigin, i);
            ufo.updateUpStreamMatrix(depOrigin);
            ufo.updateDownStreamMatrix(depOrigin);

            NextVersion nv = new NextVersion (avcFrame,
                parentFrame.drawPanel.stepID[parentFrame.drawPanel.numComponents-1][0], versionNumber );
            upstreamCalcButtonActionPerformed(avcFrame, versionNumber, i);
            downstreamCalcButtonActionPerformed(avcFrame, versionNumber, i);
        }
    }
}
}
}

```

2. Cases.GUI.DepAttributes

```

/**-----
 * class : DepAttributes -- Provides a JFrame class to get user input for
 * creating dependencies.
 * @author Art Clomera
 * @date 1/20/2003

```



```

* @compiler JDK 1.3.1
* Description: this is the gui for getting dependency attributes from the
* user. The dependency range and type functions are not currently implemented.
* side effects to class -- GraphPanel attributes:
* default value and arrayDepButtons[]
*-----
*/
package Cases.GUI;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.util.Vector;
import javax.swing.*;
import Cases.CasesFrame;
import Cases.CasesTitle;
import Cases.ComponentType;

public class DepAttributes extends JDialog implements Cases.CasesTitle, ItemListener {
    /** the selected component index variable */
    private int selectedCompIndex=-1;
    /**
     * depCompVector : contains all component type objects
     */
    public DepAttributes(Dialog owner, CasesFrame frame) {
        super (owner,"Dependency Attribute",true);
        ownerWindow=frame;
        initGUI();

        setCompComboBox(ownerWindow.psfWindow.compVector);
        this.typeChoice.add("Risk");
        this.typeChoice.add("Safety");
        this.typeChoice.add("Parent/Child");
        this.rangeChoice.add("0-9");
        this.rangeChoice.add("Boolean");
        this.rangeChoice.add("1/3/9");
    }

    /**
     * returns the dependency version
     * @return String
     */
    public String getDepVersion(){ return depVersion; }

    /**
     * method to set the dependency version
     * @param String
     */
    public void setDepVersion(String depVersion){ this.depVersion = depVersion; }

    /**
     * Refresh compHashtable and existedCompComboBox, and add new items
     */
    /**
     * @param compVector : contains the current component type objects
     */
    private void setCompComboBox(Vector compVector ){
        this.compComboBox.removeAllItems();
        this.compComboBox.addItem(COMPONENT_TYPE_TITLE);

        for( int i=0; i< compVector.size(); i++){
            ComponentType ct = (ComponentType)compVector.elementAt( i );
            String ctName = ct.getComponentName();
            Integer ctIndex = new Integer(ct.getComponentValue());
            this.compComboBox.addItem(ctName);
            this.compIndexComboBox.addItem(ctIndex);
        }
    }
}

```

```

    }
}

/**
 * a method to track item state changes for the component combo box
 */
    public void itemStateChanged( ItemEvent event) {
        Object object = event.getSource();
        if (object == compComboBox)
            compComboBox_itemStateChanged(event);
    }

/**
 * List all existing component type objects in the project
 * and allow a user to view or edit them
 */
    private void compComboBox_itemStateChanged(ItemEvent event)
    {
        if( event.getStateChange() == ItemEvent.SELECTED ) {
            String selectedCompID = (String)event.getItem();
            this.selectedCompIndex = this.compComboBox.getSelectedIndex();
        }
    }

/**
 * method to initialize GUI and its components
 */
private void initGUI() {
    saveButton = new Button();
    saveButton.setLabel("Save");
    saveButton.addActionListener(new saveButtonActionListener ());
    cancelButton = new Button();
    cancelButton.setLabel("Cancel");
    cancelButton.addActionListener(new cancelButtonActionListener ());
    compComboBox.setToolTipText("This dependency will originate from this component");
    compComboBox.setBounds(225,150,300,22);
    compComboBox.addItemListener(this);
    depValue = new TextField();
    label6 = new Label();
    rangeChoice = new Choice();
    label5 = new Label();
    typeChoice = new Choice();
    label4 = new Label();
    descriptionTextArea = new JTextArea();
    label3 = new Label();
    shortName = new TextField();
    label2 = new Label();

    getContentPane().setLayout(new java.awt.GridBagLayout());
    getContentPane().add(label8, new
java.awt.GridBagConstraints(1,8,1,1,0.0,0.0,java.awt.GridBagConstraints.CENTER,java.awt.GridBagConstraints.NONE,new
java.awt.Insets(0,0,0,0),0,0));
    getContentPane().add(saveButton, new
java.awt.GridBagConstraints(1,7,1,1,0.0,0.0,java.awt.GridBagConstraints.CENTER,java.awt.GridBagConstraints.NONE,new
java.awt.Insets(0,0,0,0),0,0));
    getContentPane().add(label1, new
java.awt.GridBagConstraints(1,6,1,1,0.0,0.0,java.awt.GridBagConstraints.CENTER,java.awt.GridBagConstraints.NONE,new
java.awt.Insets(0,0,0,0),0,0));
    getContentPane().add(label7,
new java.awt.GridBagConstraints(1, 5, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.HORIZONTAL,
new java.awt.Insets(1, 3, 2, 21), 0, 0));
    getContentPane().add(depValue,
new java.awt.GridBagConstraints(2, 4, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.HORIZONTAL,
new java.awt.Insets(1, 0, 5, 0), 0, 0));
    getContentPane().add(label6,
new java.awt.GridBagConstraints(1, 4, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.NONE,

```

```

        new java.awt.Insets(0, 0, 0, 0, 0, 0));
        getContentPane().add(rangeChoice,
        new java.awt.GridBagConstraints(2, 3, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
        java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(1, 0, 5, 0), 0, 0));
        getContentPane().add(label5,
        new java.awt.GridBagConstraints(1, 3, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
        java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(1, 0, 2, 2), 0, 0));
        getContentPane().add(typeChoice,
        new java.awt.GridBagConstraints(2, 2, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
        java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(1, 0, 6, 0), 0, 0));
        getContentPane().add(label4,
        new java.awt.GridBagConstraints(1, 2, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
        java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(1, 1, 3, 25), 0, 0));
        getContentPane().add(descriptionTextArea,
        new java.awt.GridBagConstraints(2, 1, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
        java.awt.GridBagConstraints.BOTH,
        new java.awt.Insets(3, 0, 4, 0), 0, 0));
        getContentPane().add(label3,
        new java.awt.GridBagConstraints(1, 1, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
        java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(31, 0, 33, 6), 0, 0));
        getContentPane().add(shortName,
        new java.awt.GridBagConstraints(2, 0, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
        java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(1, 0, 6, 0), 0, 0));
        getContentPane().add(label2,
        new java.awt.GridBagConstraints(1, 0, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
        java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(1, 2, 2, 21), 0, 0));
        getContentPane().add(compComboBox,
        new java.awt.GridBagConstraints(2, 5, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
        java.awt.GridBagConstraints.NONE,
        new java.awt.Insets(0, 0, 0, 0), 0, 0));
        getContentPane().add(cancelButton,
        new java.awt.GridBagConstraints(1, 9, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
        java.awt.GridBagConstraints.NONE,
        new java.awt.Insets(0, 0, 0, 0), 0, 0));
        label2.setText("Name:");
        shortName.setText("");
        label3.setText("Description:");
        descriptionTextArea.setText("");

        descriptionTextArea.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.border.BevelBorder.LOWERED));
        descriptionTextArea.setMinimumSize(new java.awt.Dimension(120,80));
        descriptionTextArea.setPreferredSize(new java.awt.Dimension(120,80));
        descriptionTextArea.setWrapStyleWord(true);
        descriptionTextArea.setLineWrap(true);
        label4.setText("Type:");
        label5.setText("Value Range:");
        label6.setText("Default Value:");
        depValue.setText("");
        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setResizable(false);
        setBounds(new java.awt.Rectangle(0, 0, 364, 369));
        setSize(new java.awt.Dimension(350, 400));
        // setState(15);
        label7.setText("Origin:");
        label11.setText("");
        label11.setFont(new java.awt.Font("Dialog", java.awt.Font.PLAIN, 7));
        label8.setText("");
        label8.setFont(new java.awt.Font("Dialog", java.awt.Font.PLAIN, 7));
        compComboBox.addActionListener(new ActionListener() {public void actionPerformed(ActionEvent
e){compComboBox(e);}});
    }

```

```

/**
 * method if cancel button is pressed
 */
public void cancelButtonActionPerformed(ActionEvent e) {
    cancelFlag=true;
    this.hide();
}

/**
 * method if save button is pressed
 */
public void saveButtonActionPerformed(ActionEvent e) {
    double amount;
    // the code accepts any double value, but future version should
    // verify that amount is within the range selected.
    try {
        amount = Double.valueOf(depValue.getText()).doubleValue();
    } catch (java.lang.NumberFormatException e2) {
        depValue.setText("Invalid Input");
        return;
    }

    if (selectedCompIndex<=0)
        JOptionPane.showMessageDialog(this,"You must select a component","Invalid Selection",JOptionPane.ERROR_MESSAGE);
    else if (amount>=0 && amount <=9) {
        JButton tempButton = new JButton();
        tempButton.setLabel(this.shortName.getText());

        int temp = ((Integer)this.compIndexComboBox.getItemAt(selectedCompIndex-1)).intValue();
        this.setDepOrigin(temp);

        setVisible( false );
    }
    else
        depValue.setText("Range 0..9");
}

/**
 * component combo box event
 */
public void compComboBox(ActionEvent e) { }

/**
 * returns the dependency origin
 * @return String
 */
public String getDepOrigin(){
    ComponentType tempCT = ((ComponentType)ownerWindow.psfWindow.compVector.get(depOrigin)) ;
    return tempCT.getComponentID();
}

/**
 * sets the dependency origin to passed parameter depOrigin
 * @param int
 */
public void setDepOrigin(int depOrigin){
    this.depOrigin = depOrigin;
}

/**
 * returns the dependency's short name
 * @return String
 */
public String getShortName () {
    return this.shortName.getText();
}

```

```

/**
 * sets the dependency's short name to inputString
 * @param String
 */
public void setShortName (String inputString) {
    this.shortName.setText(inputString);
}

/**
 * returns the dependency description
 * @return String
 */
public String getDescription () {
    return this.descriptionTextArea.getText();
}

/**
 * set the dependency description to inputString
 * @param String
 */
public void setDescription (String inputString) {
    this.descriptionTextArea.setText(inputString);
}

/**
 * returns the dependency default value
 * @return String
 */
public String getDefaultValue () {
    return this.depValue.getText();
}

/**
 * set the dependency value to inputString
 * @param String
 */
public void setDefaultValue (String inputString) {
    this.depValue.setText(inputString);
}

/**
 * returns the type selected
 * @return int
 */
public int getTypeSelected () {
    return this.typeChoice.getSelectedIndex();
}

/**
 * sets the dependency type by the value selected
 * @param int
 */
public void setTypeSelected (int selected) {
    this.typeChoice.select(selected);
}

/**
 * returns the dependency range
 * @return int
 */
public int getRangeSelected () {
    return this.rangeChoice.getSelectedIndex();
}

/**
 * sets the dependency range based on value selected
 * @param int
 */

```

```

public void setRangeSelected (int selected) {
    this.rangeChoice.select(selected);
}

/** the dependency's name */
private String depName;
public Label label2;
/** the dependency's short name */
public TextField shortName;
public Label label3;
/** a text area for the dependency description */
public JTextArea descriptionTextArea;
public Label label4;
/** dependency type drop down list */
private Choice typeChoice;
public Label label5;
/** value range drop down list */
private Choice rangeChoice;
public Label label6;
/** dependency default value */
public TextField depValue;
/** button to cancel saving the dependency */
public Button cancelButton;
/** button to save dependency */
public Button saveButton;
/** points to CasesFrame */
private CasesFrame ownerWindow;
/** a combo box of available components */
public JComboBox compComboBox = new JComboBox();
/** a combo box to relate component to an index */
public JComboBox compIndexComboBox = new JComboBox();
public Label label7 = new Label();
public Label label1 = new Label();
public Label label8 = new Label();
/** variable for dependency origin */
private int depOrigin;
/** a flag to abort saving if cancel button is pressed */
public boolean cancelFlag=false;
/** depVersion keeps track of the dependencies origin version (default=1.1)*/
private String depVersion="1.1";

/**
 * this class handles cancel button events
 */
public class cancelButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) { cancelButtonActionPerformed (e); }
}

/**
 * this class handles save button events
 */
public class saveButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) { saveButtonActionPerformed (e); }
}
}

```

3. Cases.GUI.FileSystemModel

```

/**-----
 * Filename: FileSystemModel.java
 * @Date: 10-29-2002
 * @Author: Modified by Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * TreeModel implementation using File objects as tree nodes.
 * Description: provides a tree file view of Cases directory.
 *-----

```

```

**/

package Cases.GUI;

import java.io.File;
import java.util.Iterator;
import java.util.Vector;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.TreeModel;
import javax.swing.tree.TreePath;
// Java core packages

public class FileSystemModel implements TreeModel {

    /** hierarchy root */
    private File root;

    /** Vector of TreeModelListeners */
    private Vector listeners = new Vector();

    /**
     * FileSystemModel constructor based on rootDirectory
     * @param File
     */
    public FileSystemModel( File rootDirectory ) {
        root = rootDirectory;
    }

    /**
     * get hierarchy root (root directory)
     * @return Object
     */
    public Object getRoot() {
        return root;
    }

    /**
     * get parent's child at given index from parent
     * @param Object and int
     * @return Object
     */
    public Object getChild( Object parent, int index )
    {
        // get parent File object
        File directory = ( File ) parent;

        // get list of files in parent directory
        String[] children = directory.list();

        // return File at given index and override toString
        // method to return only the File's name
        return new TreeFile( directory, children[ index ] );
    }

    /**
     * get parent's number of children from parent
     * @param Object
     * @return int
     */
    public int getChildCount( Object parent )
    {
        // get parent File object
        File file = ( File ) parent;

        // get number of files in directory
        if ( file.isDirectory() ) {

```

```

        String[] fileList = file.list();

        if ( fileList != null )
            return fileList.length;
    }

    return 0; // childCount is 0 for files
}

/**
 * return true if node is a file, false if it is a directory
 * @param Object
 * @return boolean
 */
public boolean isLeaf( Object node ) {
    File file = ( File ) node;
    return file.isFile();
}

/**
 * get numeric index of given child node from a parent
 * @param Object (x2)
 * @return int
 */
public int getIndexOfChild( Object parent, Object child ) {
    // get parent File object
    File directory = ( File ) parent;

    // get child File object
    File file = ( File ) child;

    // get File list in directory
    String[] children = directory.list();

    // search File list for given child
    for ( int i = 0; i < children.length; i++ ) {

        if ( file.getName().equals( children[ i ] ) ) {

            // return matching File's index
            return i;
        }
    }

    return -1; // indicate child index not found
} // end method getIndexOfChild

/**
 * invoked by delegate if value of Object at given
 * TreePath changes
 * @param TreePath and Object
 */
public void valueForPathChanged( TreePath path, Object value ) {
    // get File object that was changed
    File oldFile = ( File ) path.getLastPathComponent();

    // get parent directory of changed File
    String fileParentPath = oldFile.getParent();

    // get value of newFileName entered by user
    String newFileName = ( String ) value;

    // create File object with newFileName to rename oldFile
    File targetFile = new File( fileParentPath, newFileName );

    // rename oldFile to targetFile
    oldFile.renameTo( targetFile );
}

```



```

// get File object for parent directory
File parent = new File( fileParentPath );

// create int array for renamed File's index
int[] changedChildrenIndices =
    { getIndexOfChild( parent, targetFile ) };

// create Object array containing only renamed File
Object[] changedChildren = { targetFile };

// notify TreeModelListeners of node change
fireTreeNodesChanged( path.getParentPath(),
    changedChildrenIndices, changedChildren );

} // end method valueForPathChanged

/**
 * notify TreeModelListeners that children of parent at
 * given TreePath with given indices were changed
 * @param TreePath, int[], and Object[]
 */
private void fireTreeNodesChanged( TreePath parentPath, int[] indices, Object[] children ) {
    // create TreeModelEvent to indicate node change
    TreeModelEvent event = new TreeModelEvent( this,
        parentPath, indices, children );

    Iterator iterator = listeners.iterator();
    TreeModelListener listener = null;

    // send TreeModelEvent to each listener
    while ( iterator.hasNext() ) {
        listener = ( TreeModelListener ) iterator.next();
        listener.treeNodesChanged( event );
    }
} // end method fireTreeNodesChanged

/**
 * add given TreeModelListener
 * @param TreeModelListener
 */
public void addTreeModelListener( TreeModelListener listener ) {
    listeners.add( listener );
}

/**
 * remove given TreeModelListener
 * @param TreeModelListener
 */
public void removeTreeModelListener( TreeModelListener listener ) {
    listeners.remove( listener );
}

/**
 * TreeFile is a File subclass that overrides method
 * toString to return only the File name.
 */
private class TreeFile extends File {

    /**
     * TreeFile constructor based on a parent and child.
     * @param File and String
     */
    public TreeFile( File parent, String child ) {
        super( parent, child );
    }

}

/**

```

```

        * override method toString to return only the File name
        * and not the full path
        */
        public String toString() {
            return getName();
        }
    } // end inner class TreeFile
}

```

4. Cases.GUI.FileTreeModel

```

/**-----
 * @Filename: FileTreeModel.java
 * @Date: 3-7-2003
 * @Author: Arthur Clomera for NPS Thesis
 * @Compiler: JDK 1.3.1
 * Description: works with FileSystemModel to display and traverse files and directories.
 * -----
 **/

package Cases.GUI;

import java.io.File;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.TreeModel;
import javax.swing.tree.TreePath;

/**
 * The methods in this class allow the JTree component to traverse
 * the file system tree and display the files and directories.
 **/
class FileTreeModel implements TreeModel {
    /** We specify the root directory when we create the model.*/
    protected File root;

    /**
     * constructor based on root
     * @param File
     */
    public FileTreeModel(File root) { this.root = root; }

    /**
     * The model knows how to return the root object of the tree
     * @return Object
     */
    public Object getRoot() { return root; }

    /**
     * Tell JTree whether an object in the tree is a leaf
     * @return boolean
     * @param Object
     */
    public boolean isLeaf(Object node) { return ((File)node).isFile(); }

    /**
     * Tell JTree how many children a node has
     * @param Object
     * @return int
     */
    public int getChildCount(Object parent) {
        String[] children = ((File)parent).list();
        if (children == null) return 0;
        return children.length;
    }
}

```

```

/**
 * Fetch any numbered child of a node for the JTree.
 * Our model returns File objects for all nodes in the tree. The
 * JTree displays these by calling the File.toString() method.
 * @param Object and int
 * @return Object
 */
public Object getChild(Object parent, int index) {
    String[] children = ((File)parent).list();
    if ((children == null) || (index >= children.length)) return null;
    return new File((File) parent, children[index]);
}

/**
 * Figure out a child's position in its parent node.
 * @param Object (2x)
 * @return int
 */
public int getIndexOfChild(Object parent, Object child) {
    String[] children = ((File)parent).list();
    if (children == null) return -1;
    String childname = ((File)child).getName();
    for(int i = 0; i < children.length; i++) {
        if (childname.equals(children[i])) return i;
    }
    return -1;
}

/**
 * This method is invoked by the JTree only for editable trees.
 * This TreeModel does not allow editing, so we do not implement
 * this method. The JTree editable property is false by default.
 * @param TreePath and Object
 */
public void valueForPathChanged(TreePath path, Object newvalue) {}

/**
 * Since this is not an editable tree model, we never fire any events,
 * so we don't actually have to keep track of interested listeners
 */
public void addTreeModelListener(TreeModelListener l) {}

/** a method to remove tree model listener */
public void removeTreeModelListener(TreeModelListener l) {}
}

```

5. Cases.GUI.GraphPanel

```

/**-----
 * Filename: GraphPanel.java
 * @Date: 10-29-2002
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: This class is responsible for providing the user with visual
 * representation and functionality of the project schema
 * and access to QFD information.
 *-----
 */
package Cases.GUI;

import java.awt.*;
import java.awt.event.*;
import java.util.Hashtable;
import java.util.Observable;
import java.util.Observer;
import java.util.Vector;

```

```

import javax.swing.*;
import Cases.*;
import Cases.GUI.avc.AVCOpenStepFrame;
import Cases.GUI.util.Tools;
import Cases.QFD.CombinedHouseMatrix;
import Cases.QFD.ComponentQFD;
import Cases.QFD.HouseMatrix;
import Cases.QFD.MyDefaultTableModel;
import Cases.QFD.StepQFD;
import Cases.QFD.util.AVCTool;
import Cases.QFD.util.ExcelCSVLexer;
import Cases.QFD.util.ObjectFileOperations;
import Cases.GUI.util.NextVersion;

/**
 * GraphPanel.java
 *
 * @author: Arthur Clomera
 * @date: October 2002
 *
 * JPanel for editing project schema for CASES
 *
 */
public class GraphPanel extends JPanel implements CasesTitle, MouseListener, ItemListener
, MouseMotionListener, ActionListener, Observer {

/**
 * This is the main constructor for a project schema panel
 *
 * @param frame provides a link to the CasesFrame
 */
    public GraphPanel (CasesFrame frame){
        ownerWindow = frame;
        psf = ownerWindow.psfWindow;
        init();
        this.setBackground(Color.white);
        this.resize(800,600);
        this.show();
        addMouseListener (this); // Register mouse events
        addMouseMotionListener (this);
    }

    /** an array of all component locations */
    public Point[] component = new Point[MAX];
    /** an array of all component IDs */
    public String[] componentID = new String[MAX];
    /** an array of all step weights */
    public int[][] stepWeight = new int[MAX][MAX];
    /** an array of all step positions */
    public Point[][] stepPosition = new Point[MAX][MAX];
    /** an array of the startpoint component of a step */
    public Point[][] startComponent = new Point[MAX][MAX];
    /** an array of the endpoint component of a step */
    public Point[][] endComponent = new Point[MAX][MAX];
    /** an array for all step IDs */
    public String[][] stepID = new String[MAX][MAX];
    /** an array for the x direction of the step */
    private float[][] dir_x = new float[MAX][MAX];
    /** an array for the y direction of the step */
    private float[][] dir_y = new float[MAX][MAX];
    /** the name & other info of the dependency which can be passed to QFD */
    public Vector depAttrib = new Vector(MAX);
    /** a variable to store version number */
    private String versionNumber="0";
    /** a link to project schema */
    private ProjectSchemaFrame psf;
    /** a link to graph panel flags */
    public GraphPanelFlag gpf;

```

```

        /** color component to add color to graph */
        Color[] colorComponent = new Color[MAX];
        /** int to track changes for future undo function */
        int numChanged=0;
        /** int to track the number of steps */
        int step=0;
        /** a link to the popup menu */
        protected Popup popupMenu;
        // definition of several cursors
        private final Cursor DEFAULT_CURSOR = new Cursor (Cursor.DEFAULT_CURSOR);
        private final Cursor HAND_CURSOR = new Cursor (Cursor.HAND_CURSOR);
        private final Cursor MOVE_CURSOR = new Cursor (Cursor.MOVE_CURSOR);
        /** int to track the number of nodes */
        public int numComponents=0;
        /** start node of graph */
        int startgraph=0;
        int depValue = 0; // tracks dependency value
        public int hitComponent; // mouse clicked on or close to this node
        public int component1, component2; // numbers of nodes involved in current action

        Point thispoint=new Point(0,0); // current mouseposition
        Point oldpoint=new Point(0, 0); // previous position of node being moved
        /** link to CasesFrame */
        protected CasesFrame ownerWindow;

        // define several fonts
        Font roman= new Font("TimesRoman", Font.BOLD, 10);
        Font helvetica= new Font("Helvetica", Font.BOLD, 12);
        FontMetrics fmetrics = getFontMetrics(roman);
        /** int to adjust height of text on draw pane */
        int h = (int)fmetrics.getHeight()/3;

/**
 * This method set the initializes all arrays and sets the select flag.
 */
    public void init() {
        Tools tempTool = new Tools();
        for (int i=0; i<MAXCOMPONENTS; i++) {
            colorComponent[i]=Color.white;
            componentID[i] = tempTool.intToString(i)+"Component";
            for (int j=0; j<MAXCOMPONENTS; j++) {
                stepID[i][j]="s-"+tempTool.intToString(i)+tempTool.intToString(j)+"Component";
                stepWeight[i][j]=0;
            }
            gpf = new GraphPanelFlag();
        }
        // user can only select objects at the beginning
        gpf.setSelectFlag();
        colorComponent[startgraph]=Color.white;
    }

/**
 * returns true if a component is hit at location (x,y) within a dist
 * @param int, int, int
 * @return boolean
 */
    public boolean componentHit(int x, int y, int dist) {
        // checks if you hit a node with your mouseclick
        for (int i=0; i<numComponents; i++)
            if ( (x-component[i].x)*(x-component[i].x) +
                (y-component[i].y)*(y-component[i].y) < dist*dist ) {
                hitComponent = i;
                return true;
            }
        return false;
    }
}

/**

```

```

* returns true if a step is hit at location (x,y) within dist
* @param int, int, int
* @return boolean
*/
public boolean stepHit(int x, int y, int dist) {
// checks if you hit an arrow with your mouseclick
for (int i=0; i<numComponents; i++)
for (int j=0; j<numComponents; j++) {
    if ( ( stepWeight[i][j]>0 ) &&
        (Math.pow(x-stepPosition[i][j].x, 2) +
         Math.pow(y-stepPosition[i][j].y, 2) < Math.pow(dist, 2) ) ) {
        component1 = i;
        component2 = j;
        return true;
    }
}
return false;
}
/**
* remove the array index : This method is a recursive method which
* shifts all components left from the point of deletion. Shifting
* components requires changing the component ID information and the step
* ID information (e.g. if bComponent is deleted then the cComponent becomes the bComponent.
* all steps from or to the bComponent must be delete. all steps from or to the cComponent
* must be move to the new reference of bComponent.
* @param int
*/
public void removeComponent (int index){
    if ( (index>=1) && (index <= numComponents) ) {
        if (index==numComponents) { // base case: last component
            numComponents--;
            for (int i=0;i<numComponents;i++) {
                stepWeight[i][index-1]=0;
                stepWeight[index-1][i]=0;
            }
        }
        else { // recursive step copy all components & steps left
            component[index-1]=component[index]; // shift the component positions
            ((ComponentType)psf.compHashtable.get(componentID[index-1])).setComponentName(((ComponentType)psf.compHashtable.get(componentID[index])).getComponentName()); // shift the component names
            componentID[index-1]=componentID[index];
            if ((index>1) && (index < numComponents)) {
                for (int j=0;j<index-1;j++) {
                    // shift input steps left
                    stepWeight[j][index-1]=stepWeight[j][index];
                    ((StepType)psf.stepHashtable.get(stepID[j][index-1])).setStepName(
                    ((StepType)psf.stepHashtable.get(stepID[j][index])).getStepName());
                    ((StepType)psf.stepHashtable.get(stepID[j][index-1])).setStepPosition(
                    ((StepType)psf.stepHashtable.get(stepID[j][index])).getStepPosition());
                    ((StepType)psf.stepHashtable.get(stepID[j][index])).setStepName(((StepType)psf.stepHashtable.get(stepID[j][index])).getStepID());
                    stepWeight[j][index]=0;
                }
                // shift output steps left
                stepWeight[index-1][j]=stepWeight[index][j];
                ((StepType)psf.stepHashtable.get(stepID[index-1][j])).setStepName(
                ((StepType)psf.stepHashtable.get(stepID[index][j])).getStepName());
                ((StepType)psf.stepHashtable.get(stepID[index-1][j])).setStepPosition(
                ((StepType)psf.stepHashtable.get(stepID[index][j])).getStepPosition());
                ((StepType)psf.stepHashtable.get(stepID[index-1][j])).setStepName(((StepType)psf.stepHashtable.get(stepID[index][j])).getStepID());
                stepWeight[index][j]=0;
            } // end of for loop
            stepWeight[index-1][index-1]=stepWeight[index][index];
        }
        removeComponent(index+1); // remove item to right
    }
}

```

```

    } // end if
} // end of remove

/**
 * method to delete a component
 * this method is still limited by the array construct
 */
public void componentDelete() {
    psf = this.ownerWindow.psfWindow;
    // delete a node and the arrows coming into/outof the node
    Object[] options = { "Yes", "No" };
    int answer = JOptionPane.showOptionDialog(this, "Are you sure?",
    // warn the user that they are about to delete a component
    "Deleting", JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE, null, options,
options[0]);
    if( answer==0 ){ // they want to delete it.
        // mark the project schema at the hitComponent
        psf.existedCompComboBox.setSelectedItem(this.componentID[hitComponent]);
        // do the delete action
        psf.compDeleteButton_actionPerformed(null);
        // remove the component from the draw pane
        component[hitComponent]=new Point(-100, -100);

        ((ComponentType)psf.compHashtable.get(componentID[hitComponent])).setComponentName(componentID[hitComponent]);
    // initialize the steps the component used
        for (int j=0;j<numComponents;j++) {
            if (stepWeight[j][hitComponent]>0) {
                // delete input steps
                String tempStepID = this.stepID[j][hitComponent];
                stepWeight[j][hitComponent]=0;
                ((StepType)psf.stepHashtable.get(stepID[j][hitComponent])).setStepName(tempStepID);
                psf.existedStepComboBox.setSelectedItem(tempStepID);
                psf.stepDeleteButton_actionPerformed(null);
            }
            if (stepWeight[hitComponent][j] > 0) {
                // delete output steps
                String newStepID = stepID[hitComponent][j];
                stepWeight[hitComponent][j]=0;
                ((StepType)psf.stepHashtable.get(stepID[hitComponent][j])).setStepName(newStepID);
                psf.existedStepComboBox.setSelectedItem(newStepID);
                psf.stepDeleteButton_actionPerformed(null);
            }
        }
        // call this recursive method to shift the component array
        removeComponent(hitComponent+1);
    }
}

/**
 * returns the version number
 * @return String
 */
public String getVersionNumber () {
    return this.versionNumber.trim();
}

/**
 * a method to update a step between p1 and p2 with a weight w
 * on the draw pane
 * @param int, int, int
 */
public void stepUpdate(int p1, int p2, int w) {
    // make a new arrow from node p1 to p2 with weight w, or change
    // the weight of the existing arrow to w, calculate the resulting
    // position of the arrowhead
    int dx, dy;
    float l;

```

```

stepWeight[p1][p2]=w;

// direction line between p1 and p2
dx = component[p2].x-component[p1].x;
dy = component[p2].y-component[p1].y;

// distance between p1 and p2
l = (float)( Math.sqrt((float)(dx*dx + dy*dy)));
dir_x[p1][p2]=dx/l;
dir_y[p1][p2]=dy/l;

// calculate the start and endpoints of the arrow,
// adjust startpoints if there also is an arrow from p2 to p1
if (stepWeight[p2][p1]>0) {
    startComponent[p1][p2] = new Point((int)(component[p1].x-5*dir_y[p1][p2]),
                                         (int)(component[p1].y+5*dir_x[p1][p2]));
    endComponent[p1][p2] = new Point((int)(component[p2].x-5*dir_y[p1][p2]),
                                     (int)(component[p2].y+5*dir_x[p1][p2]));
}
else {
    startComponent[p1][p2] = new Point(component[p1].x, component[p1].y);
    endComponent[p1][p2] = new Point(component[p2].x, component[p2].y);
}

// range for arrowhead is not all the way to the start/endpoints
int diff_x = (int)(Math.abs(20*dir_x[p1][p2]));
int diff_y = (int)(Math.abs(20*dir_y[p1][p2]));

// calculate new x-position arrowhead
if (startComponent[p1][p2].x>endComponent[p1][p2].x) {
    stepPosition[p1][p2] = new Point(endComponent[p1][p2].x + diff_x +
                                     (Math.abs(endComponent[p1][p2].x-startComponent[p1][p2].x) - 2*diff_x )*
                                     (100-w)/100 , 0);
}
else {
    stepPosition[p1][p2] = new Point(startComponent[p1][p2].x + diff_x +
                                     (Math.abs(endComponent[p1][p2].x-startComponent[p1][p2].x) - 2*diff_x )*
                                     w/100, 0);
}

// calculate new y-position arrowhead
if (startComponent[p1][p2].y>endComponent[p1][p2].y) {
    stepPosition[p1][p2].y=endComponent[p1][p2].y + diff_y +
        (Math.abs(endComponent[p1][p2].y-startComponent[p1][p2].y) - 2*diff_y )*
        (100-w)/100;
}
else {
    stepPosition[p1][p2].y=startComponent[p1][p2].y + diff_y +
        (Math.abs(endComponent[p1][p2].y-startComponent[p1][p2].y) - 2*diff_y )*
        w/100;
}
}

/**
 * draws a step on g between components i and j
 * @param Graphics, int, and int.
 */
public void drawStep(Graphics g, int i, int j) {
    // draw arrow between node i and node j
    int x1, x2, x3, y1, y2, y3;

    // calculate arrowhead
    x1= (int)(stepPosition[i][j].x - 3*dir_x[i][j] + 6*dir_y[i][j]);
    x2= (int)(stepPosition[i][j].x - 3*dir_x[i][j] - 6*dir_y[i][j]);
    x3= (int)(stepPosition[i][j].x + 6*dir_x[i][j]);

    y1= (int)(stepPosition[i][j].y - 3*dir_y[i][j] - 6*dir_x[i][j]);
    y2= (int)(stepPosition[i][j].y - 3*dir_y[i][j] + 6*dir_x[i][j]);

```



```

y3=(int)(stepPosition[i][j].y + 6*dir_y[i][j]);

int[] arrowhead_x = { x1, x2, x3, x1 };
int[] arrowhead_y = { y1, y2, y3, y1 };

// draw arrow
g.drawLine(startComponent[i][j].x, startComponent[i][j].y, endComponent[i][j].x, endComponent[i][j].y);
g.fillPolygon(arrowhead_x, arrowhead_y, 4);

// write weight of arrow at an appropriate position
int dx = (int)(Math.abs(7*dir_y[i][j]));
int dy = (int)(Math.abs(7*dir_x[i][j]));
psf = this.ownerWindow.psfWindow;
String str = new String(((StepType)psf.stepHashtable.get(stepID[i][j])).getStepName());
g.setColor(Color.black);
if ((startComponent[i][j].x>endComponent[i][j].x) && (startComponent[i][j].y>=endComponent[i][j].y))
    g.drawString( str, stepPosition[i][j].x + dx, stepPosition[i][j].y - dy);
if ((startComponent[i][j].x>=endComponent[i][j].x) && (startComponent[i][j].y<endComponent[i][j].y))
    g.drawString( str, stepPosition[i][j].x - fmetrics.stringWidth(str) - dx ,
        stepPosition[i][j].y - dy);
if ((startComponent[i][j].x<endComponent[i][j].x) && (startComponent[i][j].y<=endComponent[i][j].y))
    g.drawString( str, stepPosition[i][j].x - fmetrics.stringWidth(str) ,
        stepPosition[i][j].y + fmetrics.getHeight());
if ((startComponent[i][j].x<=endComponent[i][j].x) && (startComponent[i][j].y>endComponent[i][j].y))
    g.drawString( str, stepPosition[i][j].x + dx,
        stepPosition[i][j].y + fmetrics.getHeight() );
}

/**
 * receive updates from observable objects for MVC pattern
 * @param Observable and Object
 */
public void update (Observable observable, Object object) {
    repaint();
}

/**
 * the method responsible for painting the graphics g (draw pane)
 * @param Graphics
 */
public void paint(Graphics g) {
    super.paint(g);

    numChanged =0;
    g.setFont(roman);
    g.setColor(Color.black);

    // draw a new arrow upto current mouse position
    if ((gpf.newStepFlag) && gpf.stepFlag)
        g.drawLine(component[component1].x, component[component1].y, thispoint.x, thispoint.y);
    if ((gpf.newDependency) && gpf.dependencyFlag)
        g.drawLine(component[component1].x, component[component1].y, thispoint.x, thispoint.y);

    // draw all arrows
    for (int i=0; i<numComponents; i++)
        for (int j=0; j<numComponents; j++){
            if (stepWeight[i][j]>0) {
                drawStep (g, i, j);
            }
        }

    // draw the nodes
    for (int i=0; i<numComponents; i++)
        if (component[i].x>0) {
            g.setColor(colorComponent[i]);
            g.fillOval(component[i].x-COMPONENTRADIX, component[i].y-COMPONENTRADIX,
                COMPONENTSIZE, COMPONENTSIZE);
        }
}

```

```

        // reflect the startnode being moved
        g.setColor(Color.white);
        if (gpf.moveStartFlag && gpf.selectFlag)
            g.fillOval(thispoint.x-COMPONENTRADIX, thispoint.y-COMPONENTRADIX,
                      COMPONENTSIZE, COMPONENTSIZE);

        g.setColor(Color.black);

        // draw black circles around nodes, write their names to the screen
        g.setFont(helvetica);
        for (int i=0; i<numComponents; i++)
            if (component[i].x>0) {
                g.setColor(Color.black);
                g.drawOval(component[i].x-COMPONENTRADIX, component[i].y-COMPONENTRADIX,
                          COMPONENTSIZE, COMPONENTSIZE);
                g.setColor(Color.red);
                psf = this.ownerWindow.psfWindow;
                g.drawString(((ComponentType)psf.compHashtable.get(componentID[i])).getComponentName(), component[i].x-14,
                           component[i].y-14);
            }
    }

    /**
     * the mouse drag event for step drawing
     */
    public void mouseDragged (MouseEvent e) {
        int x=e.getX();
        int y=e.getY();

        if (gpf.clickedFlag) {
            if (gpf.moveComponentFlag) {
                // move node and adjust arrows coming into/outof the node
                component[component1]=new Point(x, y);
                for (int i=0; i<numComponents; i++) {
                    if (stepWeight[i][component1]>0) {
                        stepUpdate (i, component1, stepWeight[i][component1]);
                    }
                    if (stepWeight[component1][i]>0) {
                        stepUpdate (component1, i, stepWeight[component1][i]);
                    }
                }

                repaint();
            }
            else if (gpf.moveStartFlag) {
                thispoint = new Point(x, y);
                repaint();
            }
        }
    }

    /**
     * the mouse moved event for moving components and drawing steps
     */
    public void mouseMoved(MouseEvent e) {
        int x = e.getX();
        int y = e.getY();
        if (gpf.selectFlag) {
            setCursor (DEFAULT_CURSOR);
            if (gpf.moveComponentFlag)
                setCursor (MOVE_CURSOR);
            else
                setCursor (HAND_CURSOR);
        } else if (gpf.stepFlag && gpf.newStepFlag) {
            thispoint = new Point(x, y);
            repaint();
        }
    }

```

```

    }
    else if (gpf.dependencyFlag && gpf.newDependency) {
        thispoint = new Point(x,y);
        repaint();
    }
}

/**
 * mouse clicked event not used but stub required.
 */
public void mouseClicked(MouseEvent e) {
    // System.out.println("Mouse Clicked!");
}

/**
 * save new components to file with default name and id
 * @param int and Point
 */
private void newComponent (int compIndex, Point compPosition) {
    psf = this.ownerWindow.psfWindow;
    String compName = this.componentID[this.numComponents-1];
    ComponentType compType = new ComponentType( compName, compName, "", compIndex, compPosition);
    compType.addObserver(this);
    psf.compVector.addElement( compType );
    //set step Hashtable
    psf.compHashtable.put( compName, compType );
    psf.compSaveButton_actionPerformed(null);
    // update display with any changes
    repaint();
}

/**
 * display step information in property window
 * for step between components c1 and c2.
 * @param int and int
 */
private void displayStepProperties(int c1, int c2) {
    psf.existedStepComboBox.setSelectedItem(this.stepID[c1][c2]);
    psf.configManagTabbedPane.setSelectedIndex(0);
    psf.setVisible(true);
}

/**
 * display a component information in property window for the comp.
 * @param int
 */
private void displayComponentProperties (int comp){
    // get the selected item
    psf.existedCompComboBox.setSelectedItem(this.componentID[comp]);
    // assign the compent position
    psf.compPosition=component[comp];
    // set the component tab to be visible
    psf.configManagTabbedPane.setSelectedIndex(1);
    psf.setVisible(true);
}

/**
 * Save a new step through the psf.
 */
private void newStep () {
    psf = this.ownerWindow.psfWindow;
    if( psf.selectedStepIndex <= 0 ){
        //must be first one add and save
        StepType stepType = new StepType( stepID[component1][component2], stepID[component1][component2], "",
component1, component2 );
        stepType.addObserver(this);
        psf.stepVector.addElement( stepType );
        psf.setListModel( psf.stepVector );
        psf.stepHashtable.put(stepID[component1][component2],stepType);
        psf.stepSaveButton_actionPerformed(null);
    }
}

```

```

    }
}

/**
 * a method to add new version QFD information based on the
 * newVersion parameter.
 * @param String
 */
public void addNewVersionQFDInformation (String newVersion) {
    psf = this.ownerWindow.psfWindow;
    String compName;
    // initialize data with a 10x10 set of info
    // this is to provide information for the user if no data is imported
    // the preference is for user to import data and not to use this default
    // data. Note: 10x10 Default data is for demonstration purposes only.
    /** no names are needed for the JTable */
    Object[] depNames = {"", "", "", "", "", "", "", "", "", ""};
    /** the default data will be size 10x10 dependency matrix */
    Object[][] depData = new Object[10][10];
    /** no names are needed for the left and top matrices */
    Object[] names = {"", "", ""};
    /** the default data will be size 10x3 for left and top matrices */
    Object[][] tableData = new Object[10][3];
    /** an array of strings to hold default artificially generated artifact names */
    String[] name = new String[10];
    // initialize the 10x10 matrix with 0's
    for (int col=0; col<10; col++)
        for (int row=0; row<10; row++)
            depData[col][row] = "0";
    MyDefaultTableModel tempTable;
    MyDefaultTableModel depTable = new MyDefaultTableModel (depData, depNames);

    // populate all component tables
    for (int i=0; i<psf.compVector.size(); i++) {
        ComponentType comp = (ComponentType)psf.compVector.get(i);
        compName = comp.getComponentID();
        for (int loop=0; loop<10; loop++) {
            String newCompName = compName.substring(0,4);
            name[loop] = newCompName + Integer.toString(loop);
        }
        for (int tableCol=0; tableCol<10; tableCol++){
            tableData[tableCol][0] = name[tableCol];
            tableData[tableCol][1] = name[tableCol];
            tableData[tableCol][2] = Integer.toString(tableCol);
        }
        ComponentType tempLeft = (ComponentType)psf.compVector.get(i);
        tempLeft.getComponentHashtable().put(newVersion, new ComponentQFD());

        // populate component table
        for (int j=0; j<MAX; j++) {
            tempTable = new MyDefaultTableModel (tableData, names);
            ((ComponentQFD)tempLeft.getComponentHashtable().get(newVersion)).addComponentTable(tempTable);
        }
        ((ComponentQFD)tempLeft.getComponentHashtable().get(newVersion)).setCompDepTable(depTable);
    }

    // populate all step tables
    for (int stepLoop=0; stepLoop<psf.stepVector.size(); stepLoop++) {
        // populate step tables
        ((StepType)psf.stepVector.get(stepLoop)).getStepQFDHashtable().put(newVersion, new StepQFD());
        ((StepQFD)((StepType)psf.stepVector.get(stepLoop)).getStepQFDHashtable().get(newVersion)).setStepTable(new
        MyDefaultTableModel(depData, depNames));
    }
    psf.setStepComboBox(psf.stepVector);
    psf.setCompComboBox(psf.compVector);
}

```

```

/**
 * the mouse pressed event
 */
public void mousePressed(MouseEvent e) {
    AVCOpenStepFrame avcFrame = new AVCOpenStepFrame(ownerWindow.projectName);
    popupMenu = new Popup(this,avcFrame);
    int x=e.getX();
    int y=e.getY();
    int clickCount = e.getClickCount();

    gpf.clickedFlag = true;
    // if right mouse button clicked then popup a menu
    if (e.isMetaDown()) {
        if (componentHit (x,y, COMPONENTSIZE)) {
            component1 = hitComponent;
            if (startgraph==component1) {
                gpf.moveStartFlag =true;
                thispoint = new Point(x,y);
                colorComponent[startgraph]=Color.white;
            }
            // enables decompose
            popupMenu.showPopupMenu(componentID[hitComponent],false,x,y);
        }
        else if (stepHit(x,y,5)) {
            // disables decompose
            popupMenu.showPopupMenu(stepID[component1][component2],true,x,y);
        }
    }
    // if double clicked than display property window
    else if ((clickCount > 1) && (!gpf.stepFlag) && (!gpf.dependencyFlag)) {
        if (componentHit (x,y, COMPONENTSIZE))
            displayComponentProperties (hitComponent);
        else if (stepHit (x,y,5))
            displayStepProperties(component1, component2);
    }

    // if left mouse button clicked and select button pressed then move object
    else if (gpf.selectFlag) {
        // move a node
        if (componentHit (x, y, COMPONENTSIZE)) {
            oldpoint = component[hitComponent];
            component1 = hitComponent;
            gpf.moveComponentFlag=true;
        }
    }

    // if left mouse button clicked and delete button pressed then delete object
    else if (gpf.deleteFlag) {
        // delete a node
        if (componentHit (x, y, COMPONENTSIZE)) {
            component1 = hitComponent;
            if (startgraph==component1) {
                gpf.moveStartFlag =true;
                thispoint = new Point(x,y);
                colorComponent[startgraph]=Color.white;
            }
            else
                gpf.deleteComponentFlag = true;
        }
    }
    repaint();
    }
    else if ((componentHit (x, y, COMPONENTSIZE)) && gpf.stepFlag) {
        // draw a new step
        if (!gpf.newStepFlag) {
            // start component
            gpf.newStepFlag = true;
            thispoint = new Point(x, y);
            component1 = hitComponent;

```

```

    }
    else {
        // end component
        gpf.newStepFlag = false;
        gpf.clickedFlag = false;
        component2 = hitComponent;
        if (component1 != component2) {
            stepUpdate (component1, component2, DEFAULT_WEIGHT);

            if (stepWeight[component2][component1]>0) {
                stepUpdate (component2, component1, stepWeight[component2][component1]);
            }
        }
        newStep();
    }
}
repaint();
}

else if (( !componentHit (x, y, MAX) && !stepHit (x, y, MAX) ) && gpf.componentFlag) {
    // draw new node
    // take the next available spot in the array
    if (numComponents < MAXCOMPONENTS) {
        component[numComponents++] = new Point(x, y);
        newComponent(numComponents-1, component[numComponents-1]);
    }
    else
        System.out.println ("maxnodes");
}
else
    // mouseclick too close to a point or arrowhead, so probably an error
    System.out.println("mouse to close to point or arrowhead.");
repaint();
}

/**
 * the mouse released event
 */
public void mouseReleased(MouseEvent e) {
    int x=e.getX();
    int y=e.getY();

    if (gpf.moveComponentFlag) {
        // move the node if the new position is not too close to
        // another node or outside of the panel

        component[component1]=new Point(0, 0);
        if ( componentHit (x, y, MAX) || (x<0) || (x>this.size().width) ||
            (y<0) || (y>this.size().height) ) {
            component[component1]=oldpoint;
            System.out.println("to close.");
        }
        else {
            component[component1]=new Point(x, y);
        }

        for (int i=0; i<numComponents; i++) {
            if (stepWeight[i][component1]>0)
                stepUpdate (i, component1, stepWeight[i][component1]);
            if (stepWeight[component1][i]>0)
                stepUpdate (component1, i, stepWeight[component1][i]);
        }
        gpf.moveComponentFlag = false;
    }
    else if (gpf.deleteComponentFlag) {
        componentDelete ();
        gpf.deleteComponentFlag = false;
    }
    else if (gpf.moveStartFlag) {
        // if new position is a node, this node becomes the startnode
    }
}

```

```

        if (componentHit (x, y, COMPONENTSIZE))
            startgraph=hitComponent;
            colorComponent[startgraph]=Color.white;
            gpf.moveStartFlag =false;
        }
        repaint();
    }

/**
 * the mouse entered event is not used
 */
public void mouseEntered(MouseEvent e) { }

/**
 * the mouse exited event is not used
 */
public void mouseExited(MouseEvent e) { }

/**
 * importCSVFile procedure is performed when popup command is selected
 * This method allow the user to import a CSV file, which can be selected from the Open file dialog window.
 * CSV files can be imported into components and will replace the left or top matrices data.
 */
public void importCSVFile () {
    Vector depNames = new Vector();
    for (int i=0;i<this.depAttrib.size();i++) {
        // populate depNames
        depNames.add(((DependencyType)depAttrib.get(i)).getName());
    }
    // The user should press the Yes button for a dependency import and the No button for a generic import.
    // It is important to import generic information prior to any dependency information,
    // thus the user should select "No" at this time.
    int depFlag = JOptionPane.showConfirmDialog(this, "Do you want to import a dependency value?",
        "Confirmation", JOptionPane.YES_NO_OPTION);
    this.depValue = 0;

    if( depFlag==0 ){ // import a dependency
        (new ListDialog(this, "Dependency List", depNames)).setVisible(true);
        ExcelCSVLexer tempShredder = new ExcelCSVLexer(depFlag);
        // if you import CSV file then you can't use the dependency default value
        ((DependencyType)depAttrib.get(this.depValue)).setNewDependency(false);
        System.out.println("GraphPanel:setting "+ this.depValue+" Dep Version:"+this.versionNumber);
        ((DependencyType)depAttrib.get(this.depValue)).setDepVersion(this.versionNumber);
        // get a copy of component's qfd information
        ((ComponentQFD)
            ((ComponentType)psf.compHashtable.get( this.componentID[hitComponent] )
        ).getComponentHashtable().get(this.versionNumber)).setComponentTable(tempShredder.getCSVTable().makeClone(),this.depValue);
        ObjectFileOperations ofo = new ObjectFileOperations();
        ofo.fileSaveActionPerformed(ownerWindow.pathName+"\\depAttrib.cfg",this.depAttrib);

    } else {
        ExcelCSVLexer loopShredder = new ExcelCSVLexer(depFlag);
        for (int i=0;i<MAXDEPENDENCIES;i++) {
            // make MAXDEPENDENCIES clones of table
            ((ComponentQFD) // get a copy of component's qfd information
            ((ComponentType)psf.compHashtable.get( this.componentID[hitComponent] )
        ).getComponentHashtable().get(this.versionNumber)).setComponentTable(loopShredder.getCSVTable().makeClone(),i);
        }
    }
    psf.compSaveButton_actionPerformed(null);
}

/**
 * importStepCSVFile procedure is performed when popup command is selected
 * this method allows the user to import a CSV file into a step
 * data will be placed in the dependency matrix for that step
 */
public void importStepCSVFile () {

```

```

        ExcelCSVLexer loopShredder = new ExcelCSVLexer();
        // make MAXDEPENDENCIES clones of table
        ((StepQFD) // get a copy of step's qfd information
        ((Hashtable)
        ((StepType)psf.stepHashtable.get(stepID[component1]][component2]
        )),getStepQFDHashtable()).get(this.versionNumber)).setStepTable(loopShredder.getStepCSVTable().makeClone());
        psf.stepSaveButton_actionPerformed(null);
    }

    /**
     * sets the dependency value with the depValue
     * @param int
     */
    public void setDepItemList (int depValue) {
        this.depValue = depValue;
    }

    /**
     * method for handling action events
     */
    public void actionPerformed (ActionEvent e) {
        AVCOpenStepFrame avcFrame = new AVCOpenStepFrame(ownerWindow.projectName);
        Tools newTool = new Tools();
        // save any old info.
        psf.compSaveButton_actionPerformed(null);
        psf.stepSaveButton_actionPerformed(null);
        // get saved data
        psf = this.ownerWindow.psfWindow;
        if (e.getSource()==popupMenu.getDecomposeMenuItem()) {
            // decompose a step or component
            ownerWindow.title = ownerWindow.DECOMPOSE_TITLE;
            ownerWindow.getVersionControl();
        }
        if (e.getSource()==popupMenu.getPropMenuItem()) {
            // display property window
            if (popupMenu.isStep) // display step properties
                displayStepProperties(component1, component2);
            else // display component properties
                displayComponentProperties(hitComponent);
        }
        if (e.getSource()==popupMenu.getDeleteMenuItem()) {
            // delete components and steps
            if (popupMenu.isStep) // future improvement to deleting steps
                System.out.println("Delete");
            else // delete a component
                componentDelete();
        }
        // loop through the Evolution Processes, steps and versions used in the popup menu
        for (int ehlcoun=1; ehlcoun<avcFrame.getEHLComboBox().getItemCount(); ehlcoun++){
            for (int stepcount=1; stepcount<avcFrame.getStepIDComboBox().getItemCount(); stepcount++){
                for (int v=0; v<avcFrame.getVersionComboBox().getItemCount(); v++) {
                    // import a CSV File
                    if (e.getSource()==popupMenu.getCSVMenuItem(ehlcoun,stepcount,v)) {
                        this.versionNumber=popupMenu.getCSVMenuItem(ehlcoun,stepcount,v).getLabel().trim();
                        if (popupMenu.isStep)
                            importStepCSVFile();
                        else
                            importCSVFile ();
                    }
                }
            }
        }
        for (int i=0; i<depAttrib.size(); i++) {
            // Display the house
            int value = Integer.parseInt(((DependencyType)depAttrib.get(i)).getDefaultValue());
            String depOrigin = ((DependencyType)depAttrib.get(i)).getOrigin();
            String name = ((DependencyType)depAttrib.get(i)).getName();
            Integer number = new Integer(value);
            if (e.getSource()==popupMenu.getQFDMenuItem(ehlcoun,stepcount,v,i)) {
                this.versionNumber=popupMenu.getQFDVersionMenu(ehlcoun,stepcount,v).getLabel().trim();
                MyDefaultTableModel leftTable=null, topTable=null, matrix=null;
            }
        }
    }

```



```

        if (popupMenu.isStep) {
            if (component2 != 0) { // get data for HOQ for a non-bridge step
                leftTable = ((ComponentQFD)
                    ((ComponentType)psf.compHashtable.get( this.componentID[component1]
                )),getComponentHashtable().get(versionNumber)).getComponentTable(i);
                topTable = ((ComponentQFD)
                    ((ComponentType)psf.compHashtable.get( this.componentID[component2]
                )),getComponentHashtable().get(versionNumber)).getComponentTable(i);
                matrix = ((StepQFD)
                    ((Hashtable)
                    ((StepType)psf.stepHashtable.get(stepID[component1][component2]
                )),getStepQFDHashtable().get(versionNumber)).getStepTable();
            } else { // get data for HOQ for a bridge step
                NextVersion nv = new NextVersion (avcFrame, stepID[component1][component2],
versionNumber);
                leftTable = ((ComponentQFD)
                    ((ComponentType)psf.compHashtable.get( this.componentID[component1]
                )),getComponentHashtable().get(versionNumber)).getComponentTable(i);
                topTable = ((ComponentQFD)
                    ((ComponentType)psf.compHashtable.get( this.componentID[component2]
                )),getComponentHashtable().get(nv.getNextVersion()).getComponentTable(i);
                matrix =
                ((StepQFD)((Hashtable)((StepType)psf.stepHashtable.get(stepID[component1][component2]
                )),getStepQFDHashtable().get(versionNumber)).getStepTable();
            }
            // allow user to edit matrix.
            matrix.setOrigin(true);

            // check to see if component1 is the origin
            if (component1==newTool.StringToInt(depOrigin))
                leftTable.setOrigin(true);
            else
                leftTable.setOrigin(false);
            // check if this is a new dependency if true display house with default value (number)
            if (((DependencyType)depAttrib.get(i)).isNewDependency() ) {
                System.out.println("GraphPanel:c1Name:" +componentID[component1]);
                // display house with default value of number
                HouseMatrix house = new HouseMatrix(ownerWindow.projectName, name+"
v."+versionNumber, i,
numComponents, depOrigin,
leftTable, topTable, matrix,
number, componentID[component1], componentID[component2], versionNumber);
                // dependency has been used or viewed therefore set false
                ((DependencyType)depAttrib.get(i)).setNewDependency(false);
                // update the dependency information to file depAttrib.cfg
                ObjectFileOperations ofo = new ObjectFileOperations();
                ofo.fileSaveActionPerformed(ownerWindow.pathName+"\\depAttrib.cfg",this.depAttrib);
            }
            else { // display house without default value
                System.out.println("GraphPanel:c1Name:" +componentID[component1]);
                HouseMatrix house = new HouseMatrix(ownerWindow.projectName, name+"
v."+versionNumber, i,
numComponents, depOrigin, leftTable, topTable, matrix,
componentID[component1], componentID[component2], versionNumber);
            }
            else { // Display the QFD concatenated house
                Vector matrixVector = new Vector();
                Vector leftVector = new Vector();
                topTable = ((ComponentQFD)((ComponentType)psf.compHashtable.get(
this.componentID[hitComponent] )),getComponentHashtable().get(versionNumber)).getComponentTable(i);
                AVCTool tempTool = new AVCTool(this.ownerWindow.projectName);
                System.out.println(this.componentID[hitComponent]);
                Vector tempVector =
tempTool.getSecondaryInputComponents(this.componentID[hitComponent]);
                System.out.println(tempVector.size());
                for (int tempLoop=0; tempLoop<tempVector.size(); tempLoop++) {
                    System.out.println("GraphPanel:"+tempVector.get(tempLoop).toString());

```

```

leftTable =
((ComponentQFD)((ComponentType)psf.compHashtable.get(tempVector.get(tempLoop).toString()
)).getComponentHashtable().get(versionNumber)).getComponentTable(i);
matrix = ((StepQFD)
((Hashtable)
((StepType)psf.stepHashtable.get("s-
"+(tempVector.get(tempLoop).toString().substring(0,1))+this.componentID[hitComponent]
)).getStepQFDHashtable().get(versionNumber)).getStepTable();
leftVector.add(leftTable);
matrixVector.add(matrix);
}
if (((DependencyType)depAttrib.get(i)).isNewDependency() ) {
// System.out.println("GraphPanel:c1Name:" +componentID[component1]);
// display house with multiple secondary input components (leftVector) with
default value number
CombinedHouseMatrix combinedhouse = new
CombinedHouseMatrix(ownerWindow.projectName, name+" v."+versionNumber, i,
numComponents, depOrigin, leftVector, topTable, matrixVector,
number, tempVector, componentID[component2], versionNumber);
// the dependency has been viewed therefore set flag to false
((DependencyType)depAttrib.get(i)).setNewDependency(false);
// dependency information must be saved with file operation to depAttrib.cfg file
ObjectFileOperations ofo = new ObjectFileOperations();
ofo.fileSaveActionPerformed(ownerWindow.pathName+"\\depAttrib.cfg",this.depAttrib);
}
else { // display house without default value
// System.out.println("GraphPanel:c1Name:" +componentID[component1]);
// display house with multiple secondary input components (leftVector) with
out default value
CombinedHouseMatrix combinedhouse = new
CombinedHouseMatrix(ownerWindow.projectName, name+" v."+versionNumber, i,
numComponents, depOrigin, leftVector, topTable, matrixVector,
tempVector, componentID[component2], versionNumber);
}
} // end of concatenated house
} // end of if
// Display the Dependency Roof
if (e.getSource()==popupMenu.getRoofMenuItem(ehlcount,stepcount,v,i)) {
this.versionNumber=popupMenu.getRoofVersionMenu(ehlcount,stepcount,v).getLabel().trim();
if (!popupMenu.isStep) {
// populate roof matrix data
MyDefaultTableModel leftTable =
((ComponentQFD)((ComponentType)psf.compHashtable.get( this.componentID[component1]
)).getComponentHashtable().get(versionNumber)).getComponentTable(i);
MyDefaultTableModel matrix =
((ComponentQFD)((ComponentType)psf.compHashtable.get( this.componentID[component1]
)).getComponentHashtable().get(versionNumber)).getCompDepTable();
// let matrix be editable
matrix.setOrigin(true);
if (((DependencyType)depAttrib.get(i)).isNewDependency() ) {
// System.out.println("GraphPanel:c1Name:" +componentID[component1]);
HouseMatrix house = new HouseMatrix(ownerWindow.projectName, name+"
v."+versionNumber, leftTable, matrix, number, componentID[component1], versionNumber);
((DependencyType)depAttrib.get(i)).setNewDependency(false);
// save changes
ObjectFileOperations ofo = new ObjectFileOperations();
ofo.fileSaveActionPerformed(ownerWindow.pathName+"\\depAttrib.cfg",this.depAttrib);
}
else {
// System.out.println("GraphPanel:c1Name:" +componentID[component1]);
// display the HOQ roof

```

```

HouseMatrix house = new HouseMatrix(ownerWindow.projectName, name+"
v."+versionNumber,
    leftTable, matrix, componentID[component1], versionNumber);
    } // end of else
    } // end of roof
} // end of if

}}}}
// save both component and step changes
psf.compSaveButton_actionPerformed(null);
psf.stepSaveButton_actionPerformed(null);
} // end of actionPerformed

/**
 * method for handling item state changes of popup menu
 * (e.g. user selection a menu item from the popup menu)
 */
public void itemStateChanged (ItemEvent e) {
    AVCOpenStepFrame avcFrame = new AVCOpenStepFrame(ownerWindow.projectName);
    for (int ehlcoun=1; ehlcoun<avcFrame.getEHLComboBox().getItemCount(); ehlcoun++){
        for (int stepcount=1; stepcount<avcFrame.getStepIDComboBox().getItemCount(); stepcount++){
            for (int v=0; v<avcFrame.getVersionComboBox().getItemCount(); v++) {
                // check if QFD version menu selected
                if (popupMenu.getQFDVersionMenu(ehlcoun,stepcount,v)!=null)
                if (popupMenu.getQFDVersionMenu(ehlcoun,stepcount,v).isSelected()) {

                    this.versionNumber=popupMenu.getQFDVersionMenu(ehlcoun,stepcount,v).getLabel().trim();
                }
            }
        }
    }

}

/**
 * method for clearing graph panel
 */
public void clear () {
    // remove graph from screen
    startgraph=0;
    numComponents=0;
    init();
    for (int i=0; i<MAXCOMPONENTS;i++) {
        component[i]=new Point(0,0);
        for (int j=0; j<MAXCOMPONENTS; j++)
            stepWeight[i][j]=0;
        }
    repaint();
}
}

```

6. Cases.GUI.GraphPanelFlag

```

/**-----
 * @Filename: GraphPanelFlag.java
 * @Date: 3-7-2003
 * @Author: Arthur Clomera for NPS Thesis
 * removes graph panel flags to reduce complexity of graphpanel class
 * @Compiler: JDK 1.3.1
 * -----
 */

package Cases.GUI;

public class GraphPanelFlag {
    public GraphPanelFlag() {
    }
}

```

```

// current action
public boolean newStepFlag = false;
public boolean newDependency = false;
public boolean moveStartFlag = false;
public boolean deleteComponentFlag = false;
public boolean moveComponentFlag = false;
public boolean clickedFlag = false;
// toolbar sets these flags when pressed
public boolean deleteFlag = false;
public boolean dependencyFlag = false;
public boolean selectFlag = false;
public boolean stepFlag = false;
public boolean componentFlag = false;
/**
 * This method set the delete flag as true and all others as false
 */
public void setDeleteFlag() {
    deleteFlag =true;
    dependencyFlag =false;
    selectFlag=false;
    componentFlag=false;
    stepFlag=false;
    newStepFlag =false;
}
/**
 * This method set the dependency flag as true and all others as false
 */
public void setDependencyFlag() {
    deleteFlag =false;
    dependencyFlag =true;
    selectFlag=false;
    componentFlag=false;
    stepFlag=false;
    newStepFlag =false;
}
/**
 * This method set the Select flag as true and all others as false
 */
public void setSelectFlag() {
    deleteFlag =false;
    dependencyFlag =false;
    selectFlag=true;
    componentFlag=false;
    stepFlag=false;
    newStepFlag =false;
}
/**
 * This method set the Step flag as true and all others as false
 */
public void setStepFlag() {
    deleteFlag =false;
    dependencyFlag =false;
    selectFlag=false;
    componentFlag=false;
    stepFlag=true;
    newStepFlag =false;
}
/**
 * This method set the Component flag as true and all others as false
 */
public void setComponentFlag() {
    deleteFlag =false;
    dependencyFlag =false;
    selectFlag=false;
    componentFlag=true;
    stepFlag=false;
    newStepFlag =false;
}

```

```
}
```

7. Cases.GUI.Popup

```
/**-----
 * Filename: Popup.java
 * @Date: 10-29-2002
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: The popup menu allows the user to access QFD, CSV Import, Project
 * Schema, and other functionality through the Draw Pane GUI
 *-----
 */

package Cases.GUI;

import javax.swing.*;
import Cases.DependencyType;
import Cases.GUI.avc.AVCOpenStepFrame;

/** popup menu to allow user to select actions based on component or step selected */
public class Popup extends JPopupMenu
{
    /** menu item to decompose components and steps */
    JMenuItem decomposeMenuItem = new JMenuItem("Decompose");
    /** menu item to delete components and steps */
    JMenuItem deleteMenuItem = new JMenuItem("Delete");
    /** menu item to edit or view properties of components and steps */
    JMenuItem propMenuItem = new JMenuItem("Properties");
    /** menu item to import a CSV file */
    JMenu importCSVMenu = new JMenu("Import CSV File");
    JMenu QFDMenu = new JMenu("QFD");
    JMenu RoofMenu = new JMenu("Roof");
    /** menu item for user defined dependencies note: hard coded for an upper limit of 3 ehl, 50 steps, 50 versions, and 50
dependencies*/
    JMenuItem[][][] QFDMenuItem = new JMenuItem[3][50][50][50];
    /** menu item for user defined dependencies note: hard coded for an upper limit of 3 ehl, 50 steps, 50 versions, and 50
dependencies*/
    JMenuItem[][][] RoofMenuItem = new JMenuItem[3][50][50][50];
    /** menu for versions limit 3 ehl, 50 steps, and 50 versions*/
    JMenu[][][] QFDVersionMenu = new JMenu[3][50][50];
    /** menu for versions limit 3 ehl, 50 steps, and 50 versions*/
    JMenu[][][] RoofVersionMenu = new JMenu[3][50][50];
    /** menu item for versions limit 3 ehl, 50 steps, and 50 versions*/
    JMenuItem[][][] CSVMenuItem = new JMenuItem[3][50][50];
    /** link back to graph panel */
    GraphPanel panel;
    /** link to AVC version information */
    AVCOpenStepFrame versionData;
    /** flag isStep */
    public boolean isStep;

    /**
     * constructor to generate a popup for the parent. The popup will need
     * to access information from avcosf about versions.
     * @param GraphPanel, AVCOpenStepFrame
     */
    public Popup (GraphPanel parent, AVCOpenStepFrame avcosf) {
        super ();
        panel = parent;
        versionData = avcosf;
        this.add (propMenuItem);
        this.addSeparator ();
        this.add (decomposeMenuItem);
        this.addSeparator ();
        this.add (deleteMenuItem);
    }
}
```

```

        this.addSeparator();
        this.add(QFDMenu);
        this.addSeparator();
        this.add(RoofMenu);
        this.addSeparator();
        this.add(importCSVMenu);
        decomposeMenuItem.addActionListener (parent);
        deleteMenuItem.addActionListener (parent);
        propMenuItem.addActionListener (parent);
        importCSVMenu.addActionListener(parent);
    }

    /** a QFD menu item */
    public JMenuItem getQFDMenuItem(int i,int j, int k, int l){ return QFDMenuItem[i][j][k][l]; }
    /** a roof menu item */
    public JMenuItem getRoofMenuItem(int i,int j, int k, int l){ return RoofMenuItem[i][j][k][l]; }
    /** a import CSV menu item */
    public JMenuItem getCSVMenuItem(int i,int j, int k){ return CSVMenuItem[i][j][k]; }
    /** a menu to hold QFD menu items (versions) */
    public JMenu getQFDVersionMenu(int i, int j, int k){ return QFDVersionMenu[i][j][k]; }
    /** a roof menu to hold roof menu items (versions)*/
    public JMenu getRoofVersionMenu(int i, int j, int k){ return RoofVersionMenu[i][j][k]; }

/**
 * get function to retrieve decompose operation
 * @param void
 * @return JMenuItem
 */
    public JMenuItem getDecomposeMenuItem () { return decomposeMenuItem; }
/**
 * get function to retrieve delete operation
 * @param void
 * @return JMenuItem
 */
    public JMenuItem getDeleteMenuItem () { return deleteMenuItem; }
/**
 * get function to retrieve property operation
 * @param void
 * @return JMenuItem
 */
    public JMenuItem getPropMenuItem () { return propMenuItem; }
/**
 * get function to retrieve import CSV file operation
 * @param void
 * @return JMenuItem
 */
    public JMenu getImportCSVMenu () { return importCSVMenu; }
/**
 * get function to retrieve QFD operation
 * @param void
 * @return JMenuItem
 */

    public void showPopupMenu (String stepIDclicked, boolean isStep, int x, int y) {
        // is this popup menu for a step?
        this.isStep = isStep;
        // only steps are allowed to be decomposed
        decomposeMenuItem.setEnabled(isStep);
        // only components are allowed to display roofs
        RoofMenu.setEnabled(!isStep);
        // only components are allowed to import CSV files
        importCSVMenu.setEnabled(!isStep);
        if (!isStep) {
            // put EHL menu info here
            for (int ehlcoun=1; ehlcoun<versionData.getEHLComboBox().getItemCount(); ehlcoun++){
                // put step name info here
                for (int stepcount=1; stepcount<versionData.getStepIDComboBox().getItemCount(); stepcount++){
                    String tempString = (String)versionData.getStepIDComboBox().getItemAt(stepcount);

```

```

        // if it is a secondary input or primary input component
        if ((stepIDClicked.equals(tempString.substring(2,3)+tempString.substring(4)) )||
            (stepIDClicked.equals(tempString.substring(3)) )) {
            importCSVMenu.removeAll();
            // put version info here
            for (int v=0; v<versionData.getVersionComboBox().getItemCount(); v++) {
                CSVMenuItem[ehlcount][stepcount][v]= new
JMenuItem(versionData.getVersionComboBox().getItemAt(v).toString());
                CSVMenuItem[ehlcount][stepcount][v].addActionListener(panel);
            importCSVMenu.add(CSVMenuItem[ehlcount][stepcount][v]);
            }
        }
    }
}

if (panel.depAttrib.size() > 0) { // must have dependencies to create menu items
QFDMenu.setEnabled(true); // enable the QFDmenu for the user to use
QFDMenu.removeAll();
    // put EHL menu info here
    for (int ehlcoun=1; ehlcoun<versionData.getEHLComboBox().getItemCount(); ehlcoun++){
        // put step name info here
        for (int stepcount=1; stepcount<versionData.getStepIDComboBox().getItemCount(); stepcount++){
            String tempString = (String)versionData.getStepIDComboBox().getItemAt(stepcount);
            // check for correct step name or component name
            if ( (stepIDClicked.equals(versionData.getStepIDComboBox().getItemAt(stepcount))) ||
                (stepIDClicked.equals(tempString.substring(2,3)+tempString.substring(4))) ||
                (stepIDClicked.equals(tempString.substring(3)) )) ){
                // put version info here

QFDMenu.removeAll();
RoofMenu.removeAll();
importCSVMenu.removeAll();
// populate the popup menus and menu items.
                for (int v=0; v<versionData.getVersionComboBox().getItemCount(); v++) {
                    // QFD menu
                    QFDVersionMenu[ehlcount][stepcount][v]= new
JMenu(versionData.getVersionComboBox().getItemAt(v).toString());
                    QFDVersionMenu[ehlcount][stepcount][v].addItemListener(panel);
                    // roof menu
                    RoofVersionMenu[ehlcount][stepcount][v]= new JMenuItem(versionData.getVersionComboBox().getItemAt(v).toString());
                    RoofVersionMenu[ehlcount][stepcount][v].addItemListener(panel);
                    for (int i=0; i<panel.depAttrib.size(); i++) {
                        // QFD and Roof dependencies
                        QFDMenuItem[ehlcount][stepcount][v][i] = new
JMenuItem(((DependencyType)panel.depAttrib.get(i)).getName());
                        RoofMenuItem[ehlcount][stepcount][v][i] = new JMenuItem(((DependencyType)panel.depAttrib.get(i)).getName());

QFDMenuItem[ehlcount][stepcount][v][i].addActionListener(panel);
RoofMenuItem[ehlcount][stepcount][v][i].addActionListener(panel);
// QFD and roof versions

QFDVersionMenu[ehlcount][stepcount][v].add(QFDMenuItem[ehlcount][stepcount][v][i]);
RoofVersionMenu[ehlcount][stepcount][v].add(RoofMenuItem[ehlcount][stepcount][v][i]);
                    }
                    // add all menu items to menus
                    QFDMenu.add(QFDVersionMenu[ehlcount][stepcount][v]);
                    RoofMenu.add(RoofVersionMenu[ehlcount][stepcount][v]);
                    CSVMenuItem[ehlcount][stepcount][v]= new JMenuItem(versionData.getVersionComboBox().getItemAt(v).toString());
                    CSVMenuItem[ehlcount][stepcount][v].addActionListener(panel);
                    importCSVMenu.add(CSVMenuItem[ehlcount][stepcount][v]);
                }
            }
        }
    }
}
else {
    QFDMenu.setEnabled(false);
}
}

```

```

        this.pack ();
        //Display the popupmenu at the position x,y
        //in the coordinate space of the component invoker.
        // SYT 12/28/99
        this.show (panel, x, y);
    }
} // End of the class Popup

```

8. Cases.GUI.SpiderMenu

```

/**-----
 * Filename: SpiderMenu.java
 * @Date: 10-29-2002
 * @Author: Hahn Le moved and modified by Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: This is the Cases menu which provides user access to
 * SPIDER functionality.
 * Modified with standard events and listeners
 *-----
 */

package Cases.GUI;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import Cases.CasesFrame;
import Cases.CasesTitle;
public class SpiderMenu extends JMenu implements CasesTitle, ActionListener {
    // main SPIDER functions
        JMenuItem editMenuItem = new JMenuItem("Edit");
        JMenuItem decomposeMenuItem = new JMenuItem("Decompose");
        JMenuItem componentContentMenuItem = new JMenuItem("Component Content");
        JMenuItem stepContentMenuItem = new JMenuItem("Step Content");
        JMenuItem traceMenuItem = new JMenuItem("Trace");
    /**
     * The main window which owns this menu.
     */
    protected CasesFrame ownerWindow;

    /**
     * constructor of menu to owner
     * @param CasesFrame
     */
    public SpiderMenu (CasesFrame owner) {
        super("Spider");
        this.setActionCommand("Editor");
        this.setMnemonic((int)'S');

        ownerWindow = owner;

        editMenuItem.setActionCommand("EditSpider");
        editMenuItem.setMnemonic((int)'E');
        this.add(editMenuItem);
        editMenuItem.addActionListener(this);

        decomposeMenuItem.setActionCommand("DecomposeSpider");
        decomposeMenuItem.setMnemonic((int)'D');
        this.add(decomposeMenuItem);
        decomposeMenuItem.addActionListener(this);

        this.addSeparator();

        componentContentMenuItem.setActionCommand("ComponentContent");
        componentContentMenuItem.setMnemonic((int)'C');

```



```

        this.add(componentContentMenuItem);
componentContentMenuItem.addActionListener(this);

        stepContentMenuItem.setActionCommand("StepContent");
        stepContentMenuItem.setMnemonic((int)'S');
        this.add(stepContentMenuItem);
        stepContentMenuItem.addActionListener(this);

        this.addSeparator();

        traceMenuItem.setActionCommand("TraceSpider");
        traceMenuItem.setMnemonic((int)'T');
        this.add(traceMenuItem);
        traceMenuItem.addActionListener(this);
    }

    /**
     * a method to handle standard event actions
     */
    public void actionPerformed(ActionEvent event) {
        Object object = event.getSource();
        if (object == editMenuItem) // edit SPIDER
            editMenuItem_actionPerformed(event);
        else if (object == decomposeMenuItem) // decompose SPIDER
            decomposeMenuItem_actionPerformed(event);
        else if (object == componentContentMenuItem) // Component Content
            componentContentMenuItem_actionPerformed(event);
        else if (object == stepContentMenuItem) // Step Content
            stepContentMenuItem_actionPerformed(event);
        else if (object == traceMenuItem) // trace spider
            traceMenuItem_actionPerformed(event);
    }

    /**
     * Get the current versionControl object and will connect to EditDecomposeFrame
     *
     * @param event : action event from Edit menu item
     */
    void editMenuItem_actionPerformed(ActionEvent event)
    {
        ownerWindow.title = EDIT_TITLE;
        ownerWindow.getVersionControl();
    }

    /**
     * Get the current versionControl object and will connect to EditDecomposeFrame
     *
     * @param event : action event from Decompose menu item
     */
    void decomposeMenuItem_actionPerformed(ActionEvent event)
    {
        ownerWindow.title = DECOMPOSE_TITLE;
        ownerWindow.getVersionControl();
    }

    /**
     * Get the current versionControl object and will connect to ComponentContentFrame
     *
     * @param event : action event from Component Content menu item
     */
    void componentContentMenuItem_actionPerformed(ActionEvent event)
    {
        ownerWindow.title = COMPONENT_CONTENT_TITLE;
        ownerWindow.getVersionControl();
    }
}

```

```

* Get the current versionControl object and will connect to StepContentFrame
*
* @param event : action event from Step Content menu item
*/
void stepContentMenuItem_actionPerformed(ActionEvent event)
{
    ownerWindow.title = STEP_CONTENT_TITLE;
    ownerWindow.getVersionControl();
}

/**
* Get the current versionControl object and will connect to TraceFrame
*
* @param event : action event from Trace menu item
*/
void traceMenuItem_actionPerformed(ActionEvent event)
{
    ownerWindow.title = TRACE_TITLE;
    ownerWindow.getVersionControl();
}
}

```

9. Cases.GUI.ToolsMenu

```

/**-----
* Filename: ToolMenu.java
* @Date: 10-29-2002
* @Author: Hahn Le moved and modified by Arthur Clomera for NPS Thesis
* Compiler: JDK 1.3.1
* Description: This is the Cases menu which provides user access to
* Tool functionality.
* Modified with standard events and listeners
*-----
**/

package Cases.GUI;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import Cases.CasesFrame;
import Cases.DeleteDialog;
import Cases.PersonnelFrame;

public class ToolsMenu extends JMenu implements ActionListener {
    // list of user tools Modified Netscape to Internet Explorer and added RequisitePro
    JMenuItem textEditorMenuItem = new JMenuItem("Text Editor");
    JMenuItem MSWordMenuItem = new JMenuItem("MS Word");
    JMenuItem MSExcelMenuItem = new JMenuItem("MS Excel");
    JMenuItem iexplorerMenuItem = new JMenuItem("Internet Explorer");
    JMenuItem capsMenuItem = new JMenuItem("CAPS");
    JMenuItem reqproMenuItem = new JMenuItem("RequisitePro");
    // list of personnel function
    JMenu personnelDataMenu = new JMenu("Personnel Data");
    JMenuItem addPersonnelMenuItem = new JMenuItem("Add");
    JMenuItem editPersonnelMenuItem = new JMenuItem("Edit");
    JMenuItem deletePersonnelMenuItem = new JMenuItem("Delete");
    JMenuItem queryMenuItem = new javax.swing.JMenuItem("Query");

    /**
    * The main window which owns this menu.
    */
    protected CasesFrame ownerWindow;

    /**

```

```

* constructor to connect this menu to the owner
* @param CasesFrame
*/
public ToolsMenu(CasesFrame owner) {

    super("Tools");
    this.setActionCommand("Tools");
    this.setMnemonic((int)'T');

    ownerWindow = owner;
    // Text Editor menu item
    textEditorMenuItem.setActionCommand("Text Editor");
    textEditorMenuItem.setMnemonic((int)'X');
    this.add(textEditorMenuItem);
    textEditorMenuItem.addActionListener(this);

    // MS Word menu item
    MSWordMenuItem.setActionCommand("MSWord");
    MSWordMenuItem.setMnemonic((int)'W');
    this.add(MSWordMenuItem);
    MSWordMenuItem.addActionListener(this);

    // MS Excell menu item
    MSExcelMenuItem.setActionCommand("MSExcel");
    MSExcelMenuItem.setMnemonic((int)'E');
    this.add(MSExcelMenuItem);
    MSExcelMenuItem.addActionListener(this);

    this.addSeparator();

    // Internet Explorer menu item
    iexplorerMenuItem.setActionCommand("IE");
    iexplorerMenuItem.setMnemonic((int)'I');
    this.add(iexplorerMenuItem);
    iexplorerMenuItem.addActionListener(this);

    // CAPS menu item (should be modified to SEATools
    capsMenuItem.setActionCommand("CAPS");
    capsMenuItem.setMnemonic((int)'C');
    this.add(capsMenuItem);
    capsMenuItem.addActionListener(this);

    // RequisitePro menu item
    reqproMenuItem.setActionCommand("REQPRO");
    reqproMenuItem.setMnemonic((int)'R');
    this.add(reqproMenuItem);
    reqproMenuItem.addActionListener(this);

    // personnel data menu
    personnelDataMenu.setActionCommand("personnel Data");
    personnelDataMenu.setMnemonic((int)'P');

    this.addSeparator();

    // add personnel menu item
    addPersonnelMenuItem.setActionCommand("Add");
    addPersonnelMenuItem.setMnemonic((int)'A');
    addPersonnelMenuItem.addActionListener(this);
    personnelDataMenu.add(addPersonnelMenuItem);

    // edit personnel menu item
    editPersonnelMenuItem.setActionCommand("Edit");
    editPersonnelMenuItem.setMnemonic((int)'E');
    editPersonnelMenuItem.addActionListener(this);
    personnelDataMenu.add(editPersonnelMenuItem);

    // delete personnel menu item
    deletePersonnelMenuItem.setActionCommand("Delete");

```

```

        deletePersonnelMenuItem.setMnemonic((int)'D');
deletePersonnelMenuItem.addActionListener(this);
personnelDataMenu.add(deletePersonnelMenuItem);

// query personnel menu item
queryMenuItem.setActionCommand("Query");
queryMenuItem.setMnemonic((int)'Q');
queryMenuItem.addActionListener(this);
personnelDataMenu.add(queryMenuItem);

// add personnel menu to tool menu
this.add(personnelDataMenu);
}

/**
 * a method to handle standard action events
 */
public void actionPerformed(ActionEvent event) {
    Object object = event.getSource();
    if (object == textEditorMenuItem) // user needs text editor
        textEditorMenuItem_actionPerformed(event);
    else if (object == MSWordMenuItem) // user needs MS Word
        MSWordMenuItem_actionPerformed(event);
    else if (object == MSExcelMenuItem) // user needs MS Excel
        MSExcelMenuItem_actionPerformed(event);
    if (object == iexplorerMenuItem) // user needs IE
        iexplorerMenuItem_actionPerformed(event);
    else if (object == capsMenuItem) // user needs SEATools
        capsMenuItem_actionPerformed(event);
    else if (object == reqproMenuItem) // user needs requisitePro
        reqproMenuItem_actionPerformed(event);
    else if (object == addPersonnelMenuItem) //user needs to add a person
        addPersonnelMenuItem_actionPerformed(event);
    else if (object == editPersonnelMenuItem) // user needs to edit personnel data
        editPersonnelMenuItem_actionPerformed(event);
    else if (object == deletePersonnelMenuItem) // user needs to delete personnel data
        deletePersonnelMenuItem_actionPerformed(event);
    else if (object == queryMenuItem) // user needs to query personnel data
        queryMenuItem_actionPerformed(event);
}

/**
 * Connect to notepad editor
 *
 * @param event : action event from Text menu item
 */
void textEditorMenuItem_actionPerformed(ActionEvent event)
{
    try {
        Runtime runtime = Runtime.getRuntime();
        runtime.exec(ownerWindow.NOTEPAD);
    }
    catch( Exception e ) {
        throw new RuntimeException(e.toString());
    }
}

/**
 * Connect to Microsoft Word editor
 *
 * @param event : action event from MS Word menu item
 */
void MSWordMenuItem_actionPerformed(ActionEvent event)
{
    try {
        Runtime runtime = Runtime.getRuntime();
        runtime.exec(ownerWindow.WINWORD);
    }
    catch( Exception e ) {

```

```

        throw new RuntimeException(e.toString());
    }
}

/**
 * Connect to Microsoft Excel editor
 *
 * @param event : action event from MS Excel menu item
 */
void MSEXcelMenuItem_actionPerformed(ActionEvent event)
{
    try {
        Runtime runtime = Runtime.getRuntime();
        runtime.exec(ownerWindow.EXCEL);
    }
    catch( Exception e ) {
        throw new RuntimeException(e.toString());
    }
}

/**
 * Connect to Netscape environment
 *
 * @param event : action event from Netscape menu item
 */
void iexplorerMenuItem_actionPerformed(ActionEvent event)
{
    try {
        Runtime runtime = Runtime.getRuntime();
        runtime.exec(ownerWindow.IEXPLORER);
    }
    catch( Exception e ) {
        throw new RuntimeException(e.toString());
    }
}

/**
 * Connect to Caps environment
 *
 * @param event : action event from Caps menu item
 */
void capsMenuItem_actionPerformed(ActionEvent event)
{
    try {
        Runtime runtime = Runtime.getRuntime();
        runtime.exec(ownerWindow.CAPS);
    }
    catch( Exception e ) {
        throw new RuntimeException(e.toString());
    }
}

/**
 * Connect to ReqPro environment
 *
 * @param event : action event from ReqPro menu item
 */
void reqproMenuItem_actionPerformed(ActionEvent event)
{
    try {
        Runtime runtime = Runtime.getRuntime();
        runtime.exec(ownerWindow.REQPRO);
    }
    catch( Exception e ) {
        throw new RuntimeException(e.toString());
    }
}

/**
 * Connect to PersonnelFrame

```

```

*
* @param event : action event from Personnel - Add menu item
*/
    void addPersonnelMenuItem_actionPerformed(ActionEvent event)
    {
        (new PersonnelFrame()).setVisible(true);
    }

/**
 * Go to JFileChooser to choose a file before connect to PersonnelFrame
 *
 * @param event : action event from Personnel - Edit menu item
 */
    void editPersonnelMenuItem_actionPerformed(ActionEvent event)
    {
        FileDialog fd = new FileDialog(ownerWindow, "Edit", FileDialog.LOAD);
        fd.setDirectory(ownerWindow.STAKEHOLDER.getAbsolutePath());
        fd.show();
        String fileName = fd.getFile();
        if( fileName != null ){
            String filePath = ownerWindow.STAKEHOLDER.getAbsolutePath()+"\\"+fileName;
            (new PersonnelFrame(filePath, "Edit")).setVisible(true);
        }
    }

/**
 * Connect to DeleteDialog and it has a Browse button to select a file
 * under stakeholder directory
 *
 * @param event : action event from Personnel - Delete menu item
 */
    void deletePersonnelMenuItem_actionPerformed(ActionEvent event)
    {
        (new DeleteDialog(ownerWindow, "Delete Personnel Data")).setVisible(true);
    }
    void queryMenuItem_actionPerformed(ActionEvent event)
    {
        FileDialog fd = new FileDialog(ownerWindow, "Query", FileDialog.LOAD);
        fd.setDirectory(ownerWindow.STAKEHOLDER.getAbsolutePath());
        fd.show();
        String fileName = fd.getFile();
        if( fileName != null ){
            String filePath = ownerWindow.STAKEHOLDER.getAbsolutePath()+fileName;
            (new PersonnelFrame(filePath, "Query")).setVisible(true);
        }
    }
}

```

C. CASES.GUI.AVC PACKAGE

1. Cases.GUI.AVC.AVCCreateStepFrame

```

/**-----
 * Filename: AVCCreateStepFrame.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1 Visual Cafe' dependencies have been removed
 * Description: This class contains the AVC Create Step functionality.
 *-----
 */
package Cases.GUI.avc;

```

```

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.*;
import java.util.Hashtable;
import java.util.StringTokenizer;
import java.util.Vector;
import javax.swing.*;
import Cases.CasesFrame;
import Cases.CasesTitle;
import Cases.Dependency;
import Cases.EHL;
import Cases.Interfaces.I_AVC;

////////////////////////////////////////////////////////////////
/**
 * Automatic Version Control - Create new version number for all steps
 *   in the current project at the same time. That means these steps always
 *   have the same number of versions.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_AVC
 */
////////////////////////////////////////////////////////////////

public class AVCCreateStepFrame extends JFrame implements CasesTitle, I_AVC, ActionListener, ItemListener
{
    /**
     * versionVector : stores all version numbers of current project
     */
    public Vector versionVector = new Vector();

    /**
     * pathName : current path name, C:\Cases\projectName
     */
    public String pathName = CASESDIRECTORY.getAbsolutePath();

    /**
     * EHLHashtable : stores all EHL objects which retrieve from loop.cfg file
     */
    public Hashtable EHLHashtable = new Hashtable();

    /**
     * depHashtable : stores all Dependency objects which retrieve from dependency.cfg file
     */
    public Hashtable depHashtable = new Hashtable();

    private CasesFrame ownerWindow;

    private void initGUI()
    {
        //{ INIT_CONTROLS
        setTitle("Automated Version Control - Create Step Version");
        getContentPane().setLayout(null);
        setSize(470,280);
        setVisible(false);
        JLabel9.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel9.setText("Evolution Process");
        getContentPane().add(JLabel9);
        JLabel9.setForeground(java.awt.Color.black);
        JLabel9.setBounds(15,70,140,22);
        currentLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        currentLabel.setText("Current Variant Number");
        getContentPane().add(currentLabel);
        currentLabel.setForeground(java.awt.Color.black);
    }
}

```

```

        currentLabel.setBounds(15,110,140,22);
        OKButton.setText("OK");
        OKButton.setActionCommand("jbutton");
        getContentPane().add(OKButton);
        OKButton.setBounds(158,210,75,22);
        cancelButton.setText("Cancel");
        cancelButton.setActionCommand("jbutton");
        getContentPane().add(cancelButton);
        cancelButton.setBounds(236,210,75,22);
        newVersionTextField.setEditable(false);
        getContentPane().add(newVersionTextField);
        newVersionTextField.setBackground(java.awt.Color.white);
        newVersionTextField.setForeground(java.awt.Color.black);
        newVersionTextField.setBounds(155,150,300,22);
        getContentPane().add(currentVariantComboBox);
        currentVariantComboBox.setBounds(155,110,300,22);
        JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel1.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
        JLabel1.setText("Create Step Version: ");
        getContentPane().add(JLabel1);
        JLabel1.setForeground(java.awt.Color.black);
        JLabel1.setFont(new Font("Dialog", Font.BOLD, 18));
        JLabel1.setBounds(10,6,190,25);
        projectLabel.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
        projectLabel.setText("Project Label");
        getContentPane().add(projectLabel);
        projectLabel.setForeground(java.awt.Color.black);
        projectLabel.setFont(new Font("Dialog", Font.BOLD, 18));
        projectLabel.setBounds(210,6,200,25);
        getContentPane().add(EHLComboBox);
        EHLComboBox.setBounds(155,70,300,22);
        newStepVersionButton.setText("New Step Version");
        newStepVersionButton.setActionCommand("New Step Version");
        getContentPane().add(newStepVersionButton);
        newStepVersionButton.setBounds(15,150,140,22);
        //}}

        //{{INIT_MENU
        //}}

        //{{REGISTER_LISTENERS

        OKButton.addActionListener(this);
        cancelButton.addActionListener(this);

        currentVariantComboBox.addItemListener(this);
        EHLComboBox.addItemListener(this);
        newStepVersionButton.addActionListener(this);
        //}}
    }

/**
 * To be called when Create Step Version Menu Item of CasesFrame
 * receives the event from user.
 * Retrieving Dependency and EHL object from dependency.cfg and loop.cfg files
 */
    public AVCCreateStepFrame( CasesFrame owner ){
        AVCOpenStepFrame avcopen = new AVCOpenStepFrame(owner.projectName);
        initGUI();
        ownerWindow = owner;
        this.projectLabel.setText( ownerWindow.projectName );
        avcopen.dispose();

        try{
            FileInputStream fileInput = new FileInputStream( pathName+"\\ "+ownerWindow.projectName+"\\dependency.cfg" );
            ObjectInputStream dep = new ObjectInputStream( fileInput );
            if( dep != null ){
                this.depHashtable = (Hashtable) dep.readObject();
            }
        }

```



```

dep.close();
fileInput.close();

fileInput = new FileInputStream( pathName+"\\ "+ownerWindow.projectName+"\\loop.cfg" );
ObjectInputStream loopIn = new ObjectInputStream( fileInput );
Vector loopVector = new Vector();
if( loopIn != null ){
    loopVector = (Vector)loopIn.readObject();
    if( loopVector.size() > 0 ){

        // Setting Evolution Process combobox
        this.setLoopNameComboBox( loopVector );
    }
}
loopIn.close();
fileInput.close();
}
catch( IOException e ){ debug("IOException: "+e); }
catch( ClassNotFoundException ex ){ debug("ClassNotFoundException: "+ex); }

try { EHLComboBox.setSelectedIndex(1); }
catch (IllegalArgumentException ex) { debug("Must lock project schema!" + ex); }
}

/**
 * this method controls the visibility of the frame based upon b
 * @param boolean
 */
public void setVisible(boolean b) {
    if (b)
        setLocation(50, 50);
    super.setVisible(b);
}

/**
 * this method overrides the super addNotify() method
 */
public void addNotify() {
    // Record the size of the window prior to calling parents addNotify.
    Dimension size = getSize();

    super.addNotify();

    if (frameSizeAdjusted)
        return;
    frameSizeAdjusted = true;

    // Adjust size of frame according to the insets and menu bar
    Insets insets = getInsets();
    javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
    int menuBarHeight = 0;
    if (menuBar != null)
        menuBarHeight = menuBar.getPreferredSize().height;
    setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
}

// Used by addNotify
boolean frameSizeAdjusted = false;

//{{DECLARE_CONTROLS
JLabel JLabel9 = new javax.swing.JLabel();
JLabel currentLabel = new javax.swing.JLabel();
JButton OKButton = new javax.swing.JButton();
JButton cancelButton = new javax.swing.JButton();
public JTextField newVersionTextField = new javax.swing.JTextField();
JComboBox currentVariantComboBox = new javax.swing.JComboBox();
JLabel JLabel1 = new javax.swing.JLabel();
JLabel projectLabel = new javax.swing.JLabel();

```

```

JComboBox EHLComboBox = new javax.swing.JComboBox();
JButton newStepVersionButton = new javax.swing.JButton();
//}}

//{{DECLARE_MENUS
//}}

public void actionPerformed(ActionEvent event)
{
    Object object = event.getSource();
    if (object == OKButton)
        OKButton_actionPerformed(event);
    else if (object == cancelButton)
        cancelButton_actionPerformed(event);
    else if (object == newStepVersionButton)
        newStepVersionButton_actionPerformed(event);
}

/**
 * Create new directory with new version number for all steps of the current project
 * @param event, occur when user press OK button
 */
public void OKButton_actionPerformed(ActionEvent event)
{
    checkPath(this.versionVector);
setVisible( false );
dispose();
}

/**
 * Exit AVCCreateStepFrame
 * @param event, occur when user press Cancel button
 */
public void cancelButton_actionPerformed(ActionEvent event)
{
setVisible( false );
dispose();
}

/**
 * Create new version number and the default version is 1.1
 * @param event, occur when user press New Version button
 */
public void newStepVersionButton_actionPerformed(ActionEvent event)
{
    if( (EHLComboBox.getItemCount() > 0) && (EHLComboBox.getSelectedIndex() > 0) ){
        if( this.currentVariantComboBox.getItemCount() == 0 ){
            newVersionTextField.setText("1.1");
        }
        else {
            createVersionNumber(this.versionVector);
        }
    }
}

public void itemStateChanged(ItemEvent event)
{
    Object object = event.getSource();
    if (object == currentVariantComboBox)
        currentVariantComboBox_itemStateChanged(event);
    else if (object == EHLComboBox)
        EHLComboBox_itemStateChanged(event);
}

/**
 * Allows a user to select all the available Evolution processes in the combobox
 * @param event, occur when user select Evolution Process

```

```

*/
    public void EHLComboBox_itemStateChanged(ItemEvent event)
    {
        if( event.getStateChange() == ItemEvent.SELECTED ){
            String selectedItem = (String)event.getItem();
            int selectedIndex = this.EHLComboBox.getSelectedIndex();

            if( selectedIndex > 0 ){
                if( this.EHLHashtable.containsKey( selectedItem ) ){
                    EHL ehl = (EHL)this.EHLHashtable.get( selectedItem );
                    this.versionVector = new Vector();
                    this.versionVector = tokenizeVector((String)ehl.getEHLPath());
                    this.setVariantComboBox( this.versionVector );
                }
            }
            else if( selectedIndex == 0 ){
                this.currentVariantComboBox.removeAllItems();
                this.newVersionTextField.setText("");
            }
        }
    }

/**
 * Allows a user to select different variants
 * @param event, occur when user select Variant number
 */
    public void currentVariantComboBox_itemStateChanged(ItemEvent event)
    {
        if( event.getStateChange() == ItemEvent.SELECTED ){
            String currentStep = ((String)event.getItem()).trim();
            newVersionTextField.setText("");
        }
    }

/**
 * Short cut to print the output
 * @param string : the output string
 */
    public void debug( String string ){
        System.out.println( string );
    }

/**
 * Tokenizing a string
 * @param string : tokenized string
 * @return v : vector of string without ","
 */
    public Vector tokenizeVector(String string){
        StringTokenizer st = new StringTokenizer( string, "," );
        Vector v = new Vector();
        while( st.hasMoreTokens() ){
            v.addElement( ((String)st.nextToken()).trim() );
        }
        return v;
    }

/**
 * Adding evolution processes into EHLComboBox
 * @param loopVector : vector of evolution processes
 */
    public void setLoopNameComboBox( Vector loopVector ){
        EHLComboBox.removeAllItems();
        EHLComboBox.addItem(PROCESS_TITLE);

        this.EHLHashtable = new Hashtable();

        for( int i=0; i< loopVector.size(); i++ ){
            EHL ehl = (EHL)loopVector.elementAt( i );

```

```

String loopName = ehl.getEHLName();

//set step Hashtable
this.EHLHashtable.put( loopName, ehl );

EHLComboBox.addItem(loopName);
}
}

/**
 * Adding variant number of the step into currentVariantComboBox
 * @param theVersionVector : vector of variant number
 */
public void setVariantComboBox( Vector theVersionVector ){
    File aFile = new File(this.pathName + "\\\" + ownerWindow.projectName, (String)theVersionVector.elementAt(0));
    if( aFile.isDirectory() ){
        String[] list = aFile.list();

        /* Clean the combo box before update it */
        this.currentVariantComboBox.removeAllItems();
        newVersionTextField.setText("");

        if( list.length > 0 ){
            Vector v = new Vector();
            for( int i=0; i<list.length; i++ ){
                String s = list[i];

                int index = s.indexOf(".");
                String sub1 = (s.substring(0,index)).trim();
                if( !v.contains(sub1) ){ v.addElement(sub1); }
            }
            if( v.size() > 0 ){
                for(int j=0; j<v.size(); j++ ){
                    currentVariantComboBox.addItem(v.elementAt(j));
                }
            }
        }
    }
}

/**
 * Create new version number for all steps in the current project
 * @param v : vector of string (e.g., 1.1, 1.2, 2.1, ...) and they
 * represent all existing version numbers
 */
public void createVersionNumber(Vector v){
    if( v.size() > 0 ){
        try{
            /* To check aFile is existing or not */
            File aFile = new File(this.pathName+"\\\"+ownerWindow.projectName,(String)v.elementAt(0));
            if( aFile.isDirectory() ){
                String[] list = aFile.list();
            }
            if( list.length > 0 ){
                int index = list[0].indexOf(".");
                String sub1 = ((String)currentVariantComboBox.getSelectedItem()).trim();
                String sub2 = list[0].substring(index+1);
                int theMax = Integer.parseInt(sub2);
                for( int i=1; i<list.length; i++ ){
                    index = list[i].indexOf(".");
                    String s = list[i].substring(0, index);
                    if( s.equals(sub1) ){
                        sub2 = list[i].substring(index+1);
                        int temp = Math.max(theMax,Integer.parseInt(sub2));
                        theMax = temp;
                    }
                }
                theMax++;
                String stepVersion = ((String)this.currentVariantComboBox.getSelectedItem()).trim()+ "."+theMax;

```

```

        newVersionTextField.setText(stepVersion);
    }
}
catch( Exception e){}
}

/**
 * Go to each step (e.g., s-I, s-C, s-R, ...) and create new directory with
 * the name is new version number
 * In CASES 2.0 step identifiers have been changed to the following format
 * s-abComponent instead of s-R (where ab captures the from and to component id)
 * @param v : vector of step names
 */
public void checkPath(Vector v){
    String currentVersion = this.newVersionTextField.getText();
    ownerWindow.drawPanel.addNewVersionQFDInformation(currentVersion.trim());
    if( v.size() > 0 ){
        File myFile = new File(this.pathName+"\\\\"+ownerWindow.projectName);
        if( myFile.exists() ) {
            String[] myList = myFile.list();
            for(int m=0; m<myList.length; m++){
                File f = new File(myFile, (String)myList[m]);
                if( f.isDirectory() ){
                    if( v.contains(f.getName()) ){
                        File theFile = new File(f,currentVersion);
                        theFile.mkdir();
                        if( !theFile.isDirectory() ){
                            theFile.mkdir();
                        }
                        if( theFile.isDirectory() ){
                            Dependency dep = (Dependency) this.depHashtable.get( f.getName() );
                            createFiles(dep, theFile);
                        }
                    }
                }
            }
        }
    }
}

/**
 * Create Component Content directory and link files inside it.
 * @param dep : Dependency object of theFile and get input.p and input.s files
 * @param theFile : current version directory
 */
public void createFiles( Dependency dep, File theFile){
    if( dep != null ){
        try{
            //Create Component Content subdirectory in the new step
            //and include txt.link, word.link, excel.link, data.link, url.link, and caps.link
            File compContent = new File(theFile,"Component Content");
            compContent.mkdir();
            if( compContent.isDirectory() ){
                for( int i=0; i<LINK_FILE_NAMES.length; i++){
                    File newFile = new File(compContent, LINK_FILE_NAMES[i]);
                    FileWriter fileWriter = new FileWriter( newFile );
                    BufferedWriter bw = new BufferedWriter( fileWriter );
                    bw.flush();
                    bw.close();
                    fileWriter.close();
                }
            }
        }
    }

    FileOutputStream fileOutput = new FileOutputStream(theFile.getAbsolutePath()+"\\input.p");
    DataOutputStream depPrimary = new DataOutputStream( fileOutput);
    if( depPrimary != null ){

```

```

        depPrimary.writeBytes(dep.getPrimaryInput());
    }
    depPrimary.flush();
    depPrimary.close();
    fileOutput.close();

    fileOutput = new FileOutputStream(theFile.getAbsolutePath()+"\\input.s");
    DataOutputStream depSecondary = new DataOutputStream( fileOutput);
    if( depSecondary != null ){
        depSecondary.writeBytes(dep.getSecondaryInput());
    }
    depSecondary.flush();
    depSecondary.close();
    fileOutput.close();
}
catch( IOException e ){ debug("dep_IOException: "+e);
}
}
}
}
}

```

2. Cases.GUI.AVC.AVCMenu

```

/**-----
 * Filename: AVCMenu.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1 Visual Cafe' dependencies have been removed
 * Description: This class contains the AVC Menu functionality.
 * Modified by removing menu functionality to this object and using standard
 * events and listeners.
 *-----
 */

package Cases.GUI.avc;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import Cases.CasesFrame;

public class AVCMenu extends JMenu implements ActionListener {
    JMenuItem AVCCreationMenuItem = new JMenuItem("Create Step Version");
    JMenuItem AVCOpenStepMenuItem = new JMenuItem("Open Step Version");
    JMenuItem AVCSplittingMenuItem = new JMenuItem("Evolution History Splitting");
    JMenuItem AVCMergingMenuItem = new JMenuItem("Evolution History Merging");
    /**
     * The main window which owns this menu.
     */
    protected CasesFrame ownerWindow;
    /** the main procedure which initializes the GUI */
    public AVCMenu (CasesFrame owner) {
        // top menu
        super("Automated Version Control");
        this.setActionCommand("Automated Version Control");
        this.setMnemonic((int)'A');
        ownerWindow = owner;
        // new step menu item
        AVCCreationMenuItem.setActionCommand("New Step");
        AVCCreationMenuItem.setMnemonic((int)'C');
        this.add(AVCCreationMenuItem);
        AVCCreationMenuItem.addActionListener(this);
        // open step menu item
        AVCOpenStepMenuItem.setActionCommand("Open Step");

```

```

        AVCOpenStepMenuItem.setMnemonic((int)'O');
        this.add(AVCOpenStepMenuItem);
        AVCOpenStepMenuItem.addActionListener(this);

        this.addSeparator();
        // splitting step menu item
        AVCSplittingMenuItem.setActionCommand("New Step");
        AVCSplittingMenuItem.setMnemonic((int)'S');
        this.add(AVCSplittingMenuItem);
        AVCSplittingMenuItem.addActionListener(this);
        // merging step menu item
        AVCMergingMenuItem.setActionCommand("New Step");
        AVCMergingMenuItem.setMnemonic((int)'M');
        this.add(AVCMergingMenuItem);
        AVCMergingMenuItem.addActionListener(this);
    }

    /**
     * procedure to process standard action events
     */
    public void actionPerformed(ActionEvent event) {
        Object object = event.getSource();
        if (object == AVCOpenStepMenuItem) // open step
            AVCOpenStepMenuItem_actionPerformed(event);
        else if (object == AVCCreationMenuItem) // create step
            AVCCreationMenuItem_actionPerformed(event);
        else if (object == AVCMergingMenuItem) // merge step
            AVCMergingMenuItem_actionPerformed(event);
        else if (object == AVCSplittingMenuItem) // split step
            AVCSplittingMenuItem_actionPerformed(event);
    }

    /**
     * Connect to AVCCreateStepFrame
     *
     * @param event : action event from Create Step Version
     */
    void AVCCreationMenuItem_actionPerformed(ActionEvent event) {
        (new AVCCreateStepFrame(ownerWindow)).setVisible(true);
    }

    /**
     * Connect to AVCOpenStepFrame
     *
     * @param event : action event from Open Step Version
     */
    void AVCOpenStepMenuItem_actionPerformed(ActionEvent event) {
        (new AVCOpenStepFrame(ownerWindow.projectName)).setVisible(true);
    }

    /**
     * Connect to AVCEHSplittingFrame
     *
     * @param event : action event from Evolution History Splitting
     */
    void AVCSplittingMenuItem_actionPerformed(ActionEvent event) {
        (new AVCSplittingFrame(ownerWindow)).setVisible(true);
    }

    /**
     * Connect to AVCEHMERGINGFrame
     *
     * @param event : action event from Evolution History Merging
     */
    void AVCMergingMenuItem_actionPerformed(ActionEvent event) {
        (new AVCMergingFrame(ownerWindow)).setVisible(true);
    }
}

```

3. Cases.GUI.AVC.AVCMergingFrame

```
/**-----
 * Filename: AVCMergingFrame.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1 Visual Cafe' dependencies have been removed
 * Description: This class contains the AVC Merging Step functionality.
 * Modified by removing Visual Cafe dependencies and using standard events and listeners.
 *-----
 */
package Cases.GUI.avc;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.ItemEvent;

import java.awt.event.ItemListener;

import java.io.*;

import java.util.Hashtable;

import java.util.StringTokenizer;

import java.util.Vector;

import javax.swing.*;

import Cases.CasesFrame;

import Cases.CasesTitle;

import Cases.Dependency;

import Cases.EHL;

import Cases.Interfaces.I_AVC;

////////////////////////////////////
/**
 * Automatic Version Control - Evolution History Merging : to merge two existing
 * versions, and create the new version.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_AVC
 */
////////////////////////////////////
public class AVCMergingFrame extends JFrame implements CasesTitle, I_AVC, ActionListener, ItemListener
{
    /**
     * pathName : current path name, e.g. C:\Cases\projectName, D:\Cases\projectName, ...
     */
    public String pathName=CASESDIRECTORY.getAbsolutePath();

    private CasesFrame ownerWindow;

    /**
     * versionVector : stores all version numbers of current project
     */
}
```



```

public Vector versionVector = new Vector();

/**
 * EHLHashtable : stores all EHL objects which retrieve from loop.cfg file
 */
    public Hashtable EHLHashtable = new Hashtable();

/**
 * depHashtable : stores all Dependency objects which retrieve from dependency.cfg file
 */
    public Hashtable depHashtable = new Hashtable();

/**
 * main procedure to initialize the GUI and Creating AVCMergingFrame
 */
    public void initGUI() {

        //{{INIT_CONTROLS

        setTitle("Automated Version Control - Evolution History Merging");
        getContentPane().setLayout(null);
        setSize(500,360);
        setVisible(false);
        JLabel9.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel9.setText("Evolution Process");
        getContentPane().add(JLabel9);
        JLabel9.setForeground(java.awt.Color.black);
        JLabel9.setBounds(26,70,145,22);
        currentStepLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        currentStepLabel.setText("Current Step Version");
        getContentPane().add(currentStepLabel);
        currentStepLabel.setForeground(java.awt.Color.black);
        currentStepLabel.setBounds(23,110,144,22);
        JLabel11.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel11.setText("Variant Type");
        getContentPane().add(JLabel11);
        JLabel11.setForeground(java.awt.Color.black);
        JLabel11.setBounds(23,190,145,22);
        OKButton.setText("OK");
        OKButton.setActionCommand("jbutton");
        getContentPane().add(OKButton);
        OKButton.setBounds(171,290,75,22);
        cancelButton.setText("Cancel");
        cancelButton.setActionCommand("jbutton");
        getContentPane().add(cancelButton);
        cancelButton.setBounds(249,290,75,22);
        getContentPane().add(currentStepComboBox);
        currentStepComboBox.setBounds(173,110,300,22);
        JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel1.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
        JLabel1.setText("Evolution History Merging: ");
        getContentPane().add(JLabel1);
        JLabel1.setForeground(java.awt.Color.black);
        JLabel1.setFont(new Font("Dialog", Font.BOLD, 18));
        JLabel1.setBounds(0,6,250,25);
        projectLabel.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
        projectLabel.setText("Project Label");
        getContentPane().add(projectLabel);
        projectLabel.setForeground(java.awt.Color.black);
        projectLabel.setFont(new Font("Dialog", Font.BOLD, 18));
        projectLabel.setBounds(250,6,250,25);
        getContentPane().add(EHLComboBox);
        EHLComboBox.setBounds(173,70,300,22);
        JLabel2.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel2.setText("Merged Step Version");
        getContentPane().add(JLabel2);
        JLabel2.setForeground(java.awt.Color.black);
        JLabel2.setBounds(23,150,145,22);
    }

```

```

        getContentPane().add(mergedStepComboBox);
        mergedStepComboBox.setBounds(173,150,300,22);
        newVersionTextField.setEditable(false);
        getContentPane().add(newVersionTextField);
        newVersionTextField.setBackground(java.awt.Color.white);
        newVersionTextField.setForeground(java.awt.Color.black);
        newVersionTextField.setBounds(173,230,300,22);
        getContentPane().add(variantComboBox);
        variantComboBox.setBounds(173,190,300,22);
        newStepVersionButton.setText("New Step Version");
        newStepVersionButton.setActionCommand("New Step Version");
        getContentPane().add(newStepVersionButton);
        newStepVersionButton.setBounds(33,230,135,22);
        //}}

        //{{INIT_MENUS
        //}}

        //{{REGISTER_LISTENERS

        OKButton.addActionListener(this);
        cancelButton.addActionListener(this);

        EHLComboBox.addItemListener(this);
        newStepVersionButton.addActionListener(this);
        currentStepComboBox.addItemListener(this);
        mergedStepComboBox.addItemListener(this);
        variantComboBox.addItemListener(this);
        //}}

        /* Adding item to variantComboBox */
        for( int i=0; i<VARIANT_TYPES.length; i++){
            variantComboBox.addItem(VARIANT_TYPES[i]);
        }
        variantComboBox.setSelectedIndex( 0 );
    }

    /**
     * To be called when Evolution History Merging Menu Item of CasesFrame
     * receives the event from user.
     * Retrieving Dependency and EHL object from dependency.cfg and loop.cfg files
     */
    public AVCMergingFrame( CasesFrame owner ){
        ownerWindow=owner;
        initGUI();
        this.projectLabel.setText(ownerWindow.projectName );

        try{
            FileInputStream fileInput = new FileInputStream( pathName+"\\\\"+ownerWindow.projectName+"\\dependency.cfg" );
            ObjectInputStream dep = new ObjectInputStream( fileInput );
            if( dep != null ){
                this.depHashtable = (Hashtable) dep.readObject();
            }
            dep.close();
            fileInput.close();
        }
        catch( IOException e ){
            debug("IOException_Dep: "+e);
        }
        catch( ClassNotFoundException ex ){
            debug("ClassNotFoundException_Dep: "+ex);
        }
    }

    try{
        FileInputStream fileInput = new FileInputStream( pathName+"\\\\"+ownerWindow.projectName+"\\loop.cfg" );
        ObjectInputStream loopIn = new ObjectInputStream( fileInput );
        if( loopIn != null ){
            Vector loopVector = new Vector();

```

```

        loopVector = (Vector)loopIn.readObject();
        if( loopVector.size() > 0){
            this.setLoopNameComboBox( loopVector );
        }
    }
    loopIn.close();
    fileInput.close();
}
catch( IOException e ){
    debug("IOException_loop: "+e);
}
catch( ClassNotFoundException ex ){
    debug("ClassNotFoundException_loop: "+ex);
}
}
}

/**
 * procedure to control visibility of frame based on b
 * @param boolean
 */
public void setVisible(boolean b) {
    if (b)
        setLocation(50, 50);
    super.setVisible(b);
}

/**
 * overrides the super method addNotify()
 */
public void addNotify() {
    // Record the size of the window prior to calling parents addNotify.
    Dimension size = getSize();

    super.addNotify();

    if (frameSizeAdjusted)
        return;
    frameSizeAdjusted = true;

    // Adjust size of frame according to the insets and menu bar
    Insets insets = getInsets();
    javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
    int menuBarHeight = 0;
    if (menuBar != null)
        menuBarHeight = menuBar.getPreferredSize().height;
    setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
}

// Used by addNotify
boolean frameSizeAdjusted = false;

//{{DECLARE_CONTROLS
JLabel JLabel9 = new javax.swing.JLabel();
JLabel currentStepLabel = new javax.swing.JLabel();
JLabel JLabel11 = new javax.swing.JLabel();
JButton OKButton = new javax.swing.JButton();
JButton cancelButton = new javax.swing.JButton();
JComboBox currentStepComboBox = new javax.swing.JComboBox();
JLabel JLabel1 = new javax.swing.JLabel();
JLabel projectLabel = new javax.swing.JLabel();
JComboBox EHLComboBox = new javax.swing.JComboBox();
JLabel JLabel2 = new javax.swing.JLabel();
JComboBox mergedStepComboBox = new javax.swing.JComboBox();
JTextField newVersionTextField = new javax.swing.JTextField();
JComboBox variantComboBox = new javax.swing.JComboBox();
JButton newStepVersionButton = new javax.swing.JButton();
//}}

//{{DECLARE_MENUS
/>
```

```

    /** procedure to process action events */
    public void actionPerformed(ActionEvent event)    {
        Object object = event.getSource();
        if (object == OKButton)
            OKButton_actionPerformed(event);
        else if (object == cancelButton)
            cancelButton_actionPerformed(event);
        else if (object == newStepVersionButton)
            newStepVersionButton_actionPerformed(event);
    }

    /**
     * Create new directory with new version number for all steps of the current project
     * @param event, occur when user press OK button
     */
    public void OKButton_actionPerformed(ActionEvent event)    {
        checkPath(this.versionVector);
        setVisible( false );
        dispose();
    }

    /**
     * Exit AVCMergingFrame
     * @param event, occur when user press Cancel button
     */
    public void cancelButton_actionPerformed(ActionEvent event)    {
        setVisible( false );
        dispose();
    }

    /**
     * Create new version number and the default version is 1.1
     * @param event, occur when user press New Version button
     */
    public void newStepVersionButton_actionPerformed(ActionEvent event)    {
        if( (EHLComboBox.getItemCount() > 0) && (EHLComboBox.getSelectedIndex() > 0) ){
            if( this.currentStepComboBox.getItemCount()>0 &&
                this.mergedStepComboBox.getItemCount() > 0 ){
                createVersionNumber(variantComboBox.getSelectedIndex());
            }
            else{
                newVersionTextField.setText("1.1");
            }
        }
    }

    public void itemStateChanged(ItemEvent event)
    {
        Object object = event.getSource();
        if (object == EHLComboBox)
            EHLComboBox_itemStateChanged(event);
        else if (object == currentStepComboBox)
            currentStepComboBox_itemStateChanged(event);
        else if (object == mergedStepComboBox)
            currentStepComboBox_itemStateChanged(event);
        else if (object == variantComboBox)
            currentStepComboBox_itemStateChanged(event);
    }

    /**
     * Allows a user to select all the available Evolution processes in the combobox
     * @param event, occur when user select Evolution Process
     */
    public void EHLComboBox_itemStateChanged(ItemEvent event)
    {
        if( event.getStateChange() == ItemEvent.SELECTED ){
            String selectedItem = (String)event.getItem();

```

```

int selectedIndex = this.EHLComboBox.getSelectedIndex();

this.currentStepComboBox.removeAllItems();
this.mergedStepComboBox.removeAllItems();
newVersionTextField.setText("");

if( selectedIndex > 0 ){
    if( this.EHLHashtable.containsKey( selectedltem ) ){
        EHL ehl = (EHL)this.EHLHashtable.get( selectedltem );
        this.versionVector = new Vector();
        this.versionVector = tokenizeVector((String)ehl.getEHLPath());
        this.setVersionComboBoxes( this.versionVector );
    }
}
}
}

/**
 * Allows a user to select all the available steps in the combobox
 * @param event, occur when user select currentStepComboBox
 */
public void currentStepComboBox_itemStateChanged(ItemEvent event)
{
    if( event.getStateChange() == ItemEvent.SELECTED ){
        newVersionTextField.setText("");
    }
}

/**
 * Short cut to print the output
 * @param string : the output string
 */
public void debug( String string ){
    System.out.println( string );
}

/**
 * Adding evolution processes into EHLComboBox
 * @param loopVector : vector of evolution processes
 */
public void setLoopNameComboBox( Vector loopVector ){
    EHLComboBox.removeAllItems();
    EHLComboBox.addItem(PROCESS_TITLE);

    this.EHLHashtable = new Hashtable();

    for( int i=0; i< loopVector.size(); i++ ){
        EHL ehl = (EHL)loopVector.elementAt( i );
        String loopName = ehl.getEHLName();

        //set step Hashtable
        this.EHLHashtable.put( loopName, ehl );

        this.EHLComboBox.addItem(loopName);
    }
}

/**
 * Adding version numbers to currentStepComboBox and mergedStepComboBox
 * @param versionVector : vector of version numbers
 */
public void setVersionComboBoxes( Vector versionVector ){

    File aFile = new File(this.pathName + "\\\" + ownerWindow.projectName, (String)versionVector.elementAt(0));
    if( aFile.isDirectory() ){
        String[] list = aFile.list();

        if( list.length > 0 ){

```

```

        for( int i=0; i<list.length; i++ ){
            this.currentStepComboBox.addItem(((String)list[i]).trim());
            this.mergedStepComboBox.addItem(((String)list[i]).trim());
        }
    }
}

/**
 * Tokenizing a string
 * @param string : tokenized string
 * @return v : vector of string without ","
 */
public Vector tokenizeVector(String string){
    StringTokenizer st = new StringTokenizer( string, "," );
    Vector v = new Vector();
    while( st.hasMoreTokens() ){
        v.addElement( ((String)st.nextToken()).trim() );
    }
    return v;
}

/**
 * Create new version number for all steps in the current project
 * @param typeIndex = 0 ==> variant is old, new version = currentVariant.(highestVersion+1);
 *               = 1 ==> variant is new, new version = (highestVariant+1).(highestVersion+1);
 */
public void createVersionNumber(int typeIndex){
    String current = ((String)currentStepComboBox.getSelectedItem()).trim();
    String merged = ((String)mergedStepComboBox.getSelectedItem()).trim();

    if( current.equals(merged) ){
        JOptionPane.showMessageDialog(this, " Current Step Version must be different from Merged Step Version!",
            "Error Message", JOptionPane.ERROR_MESSAGE);
        return;
    }
    else{
        try{
            int index1 = current.indexOf(".");
            int index2 = merged.indexOf(".");
            String sub1 = current.substring(index1+1);
            String sub2 = merged.substring(index2+1);
            int theMax2 = Math.max(Integer.parseInt(sub1),Integer.parseInt(sub2));
            theMax2++;
            String mergedVersion = null;

            if( typeIndex == 0 ){
                mergedVersion = current.substring(0,index1)+"."+theMax2;
            }
            else if( typeIndex == 1 ){
                index1 = current.indexOf(".");
                index2 = merged.indexOf(".");
                sub1 = current.substring(0, index1);
                sub2 = merged.substring(0, index2);
                int theMax1 = Math.max(Integer.parseInt(sub1),Integer.parseInt(sub2));
                theMax1++;
                mergedVersion = theMax1+"."+theMax2;
            }
        }

        for( int i=0; i<currentStepComboBox.getItemCount(); i++ ){
            String s = (String)currentStepComboBox.getItemAt(i);
            if( s.equals(mergedVersion) ){
                JOptionPane.showMessageDialog(this, mergedVersion+" version already exists in this project!",
                    "Error Message", JOptionPane.ERROR_MESSAGE);
                return;
            }
        }
        else if( i==currentStepComboBox.getItemCount()-1 ){
            newVersionTextField.setText(mergedVersion);
        }
    }
}

```

```

    }
    }
    }
    catch( Exception e){}
}
}

/**
 * Go to each step (e.g., s-I, s-C, s-R, ...) and create new directory with
 * the name is new version number
 * @param v : vector of step names
 */
public void checkPath(Vector v){
    String mergedVersion = this.newVersionTextField.getText();
    ownerWindow.drawPanel.addNewVersionQFDInformation(mergedVersion.trim());
    if( v.size() > 0 ){
        File myFile = new File(this.pathName + "\\\" + ownerWindow.projectName);
        if( myFile.exists() ) {
            String[] myList = myFile.list();
            for(int m=0; m<myList.length; m++){
                File f = new File(myFile, (String)myList[m]);
                if( f.isDirectory() ){
                    if( v.contains(f.getName()) ){
                        File theFile = new File(f,mergedVersion);
                        theFile.mkdir();
                        if( !theFile.isDirectory() ){
                            theFile.mkdir();
                        }
                    }
                    if( theFile.isDirectory() ){
                        Dependency dep = (Dependency) this.depHashtable.get( f.getName() );
                        createFiles(dep, theFile);
                    }
                }
            }
        }
    }
}

/**
 * Create Component Content directory and link files inside it.
 * @param dep : Dependency object of theFile and get input.p and input.s files
 * @param theFile : current version directory
 */
public void createFiles( Dependency dep, File theFile){
    if( dep != null ){
        try{
            //Create Component Content subdirectory in thte new step
            //and include txt.link, data.link, url.link, and caps.link
            File compContent = new File(theFile,"Component Content");
            compContent.mkdir();
            if( compContent.isDirectory() ){
                for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
                    File newFile = new File(compContent, LINK_FILE_NAMES[i]);
                    FileWriter fileWriter = new FileWriter( newFile );
                    BufferedWriter bw = new BufferedWriter( fileWriter );
                    bw.flush();
                    bw.close();
                    fileWriter.close();
                }
            }

            FileOutputStream fileOutput = new FileOutputStream(theFile.getAbsolutePath()+"\\input.p");
            DataOutputStream depPrimary = new DataOutputStream( fileOutput);
            if( depPrimary != null ){
                depPrimary.writeBytes(dep.getPrimaryInput());
            }
            depPrimary.flush();
        }
    }
}

```

```

        depPrimary.close();
        fileOutput.close();

        fileOutput = new FileOutputStream(theFile.getAbsolutePath()+"\\input.s");
        DataOutputStream depSecondary = new DataOutputStream( fileOutput);
        if( depSecondary != null ){
            depSecondary.writeBytes(dep.getSecondaryInput());
        }
        depSecondary.flush();
        depSecondary.close();
        fileOutput.close();
    }
    catch( IOException e ){
        debug("dep_IOException: "+e);
    }
}
}
}

```

4. Cases.GUI.AVC.AVCOpenStepFrame

```

/**-----
 * Filename: AVCOpenStepFrame.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1
 * Description: This class contains the AVC Open Step functionality.
 * Modified by removing Visual Cafe dependencies and using standard events and listeners.
 *-----
 */
package Cases.GUI.avc;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.*;
import java.util.Hashtable;
import java.util.StringTokenizer;
import java.util.Vector;
import javax.swing.*;
import Cases.CasesTitle;
import Cases.EHL;
import Cases.Interfaces.I_AVCOpenStep;
import Cases.VersionControl;

/** Automatic Version Control - Open existing version number for all steps
 *   in the current project.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_AVCOpenStep
 */

public class AVCOpenStepFrame extends JFrame implements CasesTitle, I_AVCOpenStep, ActionListener, ItemListener
{
    /**
     * pathName : current path name, e.g. C:\Cases\projectName, D:\Cases\projectName, ...
     */
    public String pathName = CASESDIRECTORY.getAbsolutePath();

    /** variable to store project name */
    private String projectName;

    /**

```



```

* EHLHashtable : stores all EHL objects which retrieve from loop.cfg file
*/
    public Hashtable EHLHashtable = new Hashtable();

    /**
     * oldVersionControl : VersionControl object, keeping the content of current.cfg file
     */
    protected VersionControl oldVersionControl = null;

    /** central procedure to initialize GUI and GUI components */
    private void initGUI()
    {
        //{{INIT_CONTROLS
        setTitle("Automated Version Control - Open Step Version");
        getContentPane().setLayout(null);
        setSize(450,290);
        setVisible(false);
        JLabel9.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel9.setText("Evolution Process");
        getContentPane().add(JLabel9);
        JLabel9.setForeground(java.awt.Color.black);
        JLabel9.setBounds(5,70,110,22);
        JLabel10.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel10.setText("Step Type");
        getContentPane().add(JLabel10);
        JLabel10.setForeground(java.awt.Color.black);
        JLabel10.setBounds(5,110,110,22);
        JLabel12.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel12.setText("Step Version");
        getContentPane().add(JLabel12);
        JLabel12.setForeground(java.awt.Color.black);
        JLabel12.setBounds(5,150,110,22);
        OKButton.setText("OK");
        OKButton.setActionCommand("jbutton");
        getContentPane().add(OKButton);
        OKButton.setBounds(148,210,75,22);
        cancelButton.setText("Cancel");
        cancelButton.setActionCommand("jbutton");
        getContentPane().add(cancelButton);
        cancelButton.setBounds(226,210,75,22);
        getContentPane().add(stepIDComboBox);
        stepIDComboBox.setBounds(117,110,300,22);
        JLabel11.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        JLabel11.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
        JLabel11.setText("Open Step Version: ");
        getContentPane().add(JLabel11);
        JLabel11.setForeground(java.awt.Color.black);
        JLabel11.setFont(new Font("Dialog", Font.BOLD, 18));
        JLabel11.setBounds(0,6,180,25);
        projectLabel.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
        projectLabel.setText("Project Label");
        getContentPane().add(projectLabel);
        projectLabel.setForeground(java.awt.Color.black);
        projectLabel.setFont(new Font("Dialog", Font.BOLD, 18));
        projectLabel.setBounds(187,6,200,25);
        getContentPane().add(EHLComboBox);
        EHLComboBox.setBounds(117,70,300,22);
        getContentPane().add(versionComboBox);
        versionComboBox.setBounds(117,150,300,22);
        //}}

        //{{INIT_MENU
        //}}

        //{{REGISTER_LISTENERS

        OKButton.addActionListener(this);
        cancelButton.addActionListener(this);

```

```

        EHLComboBox.addItemListener(this);
        stepIDComboBox.addItemListener(this);
        //}}
    }

/**
 * To be called when Open Step Version Menu Item of CasesFrame
 * receives the event from user.
 * Retrieving VersionControl and EHL object from current.cfg and loop.cfg files
 */
public AVCOpenStepFrame( String projectName ){
    this.projectName = projectName;
    initGUI();
    this.projectLabel.setText( projectName );

    try{
        FileInputStream fileInput = new FileInputStream( pathName+"\\\\"+this.projectName+"\\loop.cfg" );
        ObjectInputStream loopIn = new ObjectInputStream( fileInput );
        if( loopIn != null ){
            Vector loopVector = new Vector();
            loopVector = (Vector)loopIn.readObject();
            if( loopVector.size() > 0 ){
                this.setLoopNameComboBox( loopVector );
            }
        }
        loopIn.close();
        fileInput.close();
    }
    catch( IOException e ){
        debug("IOException_loop: "+e);
    }
    catch( ClassNotFoundException ex ){
        debug("ClassNotFoundException_loop: "+ex);
    }
}

try{
    // System.out.print("AVCOpen:current.vsn: ");System.out.println(this.pathName+"\\\\"+this.projectName+"\\current.vsn");
    FileInputStream fileInput = new FileInputStream( this.pathName+"\\\\"+this.projectName+"\\current.vsn" );
    ObjectInputStream currentIn = new ObjectInputStream( fileInput );
    if( currentIn != null ){
        VersionControl oldVersionControl = (VersionControl)currentIn.readObject();
        if( oldVersionControl != null ){
            this.setInitialFrame(oldVersionControl);
        }
    }
    currentIn.close();
    fileInput.close();
}
catch( IOException e ){
    debug("IOException_currentIn: "+e);
}
catch( ClassNotFoundException ex ){
    debug("ClassNotFoundException_currentLoop: "+ex);
}
try { // always start on first elements
    EHLComboBox.setSelectedIndex(1);
    stepIDComboBox.setSelectedIndex(1);
    // versionComboBox.setSelectedIndex(0);
}
catch (IllegalArgumentException ex) {
    debug("EHL not defined yet:" + ex);
}

}

/*      public AVCOpenStepFrame(String sTitle) {setTitle(sTitle);} */
/**

```

```

* procedure to control visibility of frame based on b
* @param boolean
*/
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

    /** overrides super method addNotify() */
    public void addNotify() {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    /** {DECLARE_CONTROLS
    JLabel JLabel9 = new javax.swing.JLabel();
    JLabel JLabel10 = new javax.swing.JLabel();
    JLabel JLabel12 = new javax.swing.JLabel();
    JButton OKButton = new javax.swing.JButton();
    /** a cancel button */
    JButton cancelButton = new javax.swing.JButton();
    JLabel JLabel11 = new javax.swing.JLabel();
    /** a label with the project name */
    JLabel projectLabel = new javax.swing.JLabel();
    // combo boxes for evolution processs, versions, and step ids
    JComboBox EHLComboBox = new JComboBox();
    JComboBox versionComboBox = new JComboBox();
    JComboBox stepIDComboBox = new JComboBox();
    /**
    * returns the stepIDComboBox with step ID information
    * @return JComboBox
    */
    public JComboBox getStepIDComboBox () { return this.stepIDComboBox;}
    /**
    * returns the versionComboBox with version information
    * @return JComboBox
    */
    public JComboBox getVersionComboBox () { return this.versionComboBox; }
    /**
    * returns EHLComboBox with Evolution Hyperlink information
    * @return JComboBox
    */
    public JComboBox getEHLComboBox() { return this.EHLComboBox; }

    /**
    * procedure to handle standard action events
    */
    public void actionPerformed(ActionEvent event) {
        Object object = event.getSource();

```

```

        if (object == OKButton)
            OKButton_actionPerformed(event);
        else if (object == cancelButton)
            cancelButton_actionPerformed(event);
    }

/**
 * Create new directory with new version number for all steps of the current project
 * @param event, occur when user press OK button
 */
public void OKButton_actionPerformed(ActionEvent event)
{
    String currentLoop = (String)this.EHLComboBox.getSelectedItemAt();
    String currentStep = (String)this.stepIDComboBox.getSelectedItemAt();
    String currentVersion = (String) this.versionComboBox.getSelectedItemAt();
    String currentStatus = "";

    VersionControl vc = new VersionControl( currentLoop, currentStep,
        currentVersion, currentStatus );

    try {
        // System.out.print("AVCOpen:current.vsn: ");System.out.println(this.pathName+"\\ "+this.projectName+"\\current.vsn");
        FileOutputStream fileOutput = new FileOutputStream( this.pathName+"\\ "+this.projectName+"\\current.vsn" );
        ObjectOutputStream currentOut= new ObjectOutputStream( fileOutput );
        if( currentOut != null ){
            currentOut.writeObject( vc );
        }
        currentOut.flush();
        currentOut.close();
        fileOutput.close();
    }
    catch( IOException e1 ){
        debug("IOException_currentOut: "+e1);
    }
}

setVisible( false );
dispose();
}

/**
 * Exit AVCOpenStepFrame
 * @param event, occur when user press Cancel button
 */
public void cancelButton_actionPerformed(ActionEvent event) {
    setVisible( false );
    dispose();
}

/** procedure to check if item states have changed in combo boxes */
public void itemStateChanged(ItemEvent event) {
    Object object = event.getSource();
    if (object == EHLComboBox)
        EHLComboBox_itemStateChanged(event);
    else if (object == stepIDComboBox)
        stepIDComboBox_itemStateChanged(event);
}

/**
 * Allows a user to select all the available Evolution processes in the combobox
 * @param event, occur when user select Evolution Process
 */
public void EHLComboBox_itemStateChanged(ItemEvent event) {
    if( event.getStateChange() == ItemEvent.SELECTED ){
        String selectedItem = (String)event.getItem();
        int selectedIndex = this.EHLComboBox.getSelectedIndex();

        if( selectedIndex > 0 ){
            if( this.EHLHashtable.containsKey( selectedItem ) ){

```

```

        EHL ehl = (EHL)this.EHLHashtable.get( selectedItem );

        StringTokenizer st = new StringTokenizer( (String)ehl.getEHLPath(), "," );
        Vector tokenizeVector = new Vector();
        while( st.hasMoreTokens() ){
            tokenizeVector.addElement( st.nextToken() );
        }
        this.setStepComboBox( tokenizeVector );
    }
}
else if( selectedIndex == 0 ){
    this.stepIDComboBox.removeAllItems();
    this.versionComboBox.removeAllItems();
}
}
}

/**
 * Allows a user to select all the available step ID in the combobox
 * @param event, occur when user select step ID
 */
public void stepIDComboBox_itemStateChanged(ItemEvent event) {
    if( event.getStateChange() == ItemEvent.SELECTED ){
        String selectedStep = ((String)event.getItem()).trim();
        int selectedIndex = this.stepIDComboBox.getSelectedIndex();

        if( selectedIndex > 0 ){

            Vector versionVector = new Vector();
            File existedDir = new File( this.pathName + "\\\" + this.projectName, selectedStep );
            if( existedDir.isDirectory() ){
                String[] fileList = existedDir.list();

                for( int j=0; j<fileList.length; j++ ){
                    versionVector.addElement( fileList[j] );
                }
                this.setVersionComboBox(versionVector);
            }
        }
        else if( selectedIndex == 0 ){
            this.versionComboBox.removeAllItems();
        }
    }
}

/**
 * Short cut to print the output
 * @param string : the output string
 */
public void debug( String string ){
    System.out.println( string );
}

/**
 * Adding evolution processes into EHLComboBox
 * @param loopVector : vector of evolution processes
 */
public void setLoopNameComboBox( Vector loopVector ){
    EHLComboBox.removeAllItems();
    EHLComboBox.addItem(PROCESS_TITLE);

    this.EHLHashtable = new Hashtable();

    for( int i=0; i<loopVector.size(); i++ ){
        EHL ehl = (EHL)loopVector.elementAt( i );

        //set step Hashtable
        this.EHLHashtable.put( ehl.getEHLName(), ehl );
    }
}

```

```

        this.EHLComboBox.addItem(ehl.getEHLName());
    }
}

/**
 * Adding step ID into stepIDComboBox
 * @param stepVector : vector of step ID
 */
public void setStepComboBox( Vector stepVector ){
    this.stepIDComboBox.removeAllItems();
    this.stepIDComboBox.addItem(STEP_TYPE_TITLE);

    Vector versionVector = new Vector();
    for( int i=0; i< stepVector.size(); i++ ){
        String stepID = (String)stepVector.elementAt( i );
        this.stepIDComboBox.addItem( stepID.trim() );
    }

    this.setVersionComboBox( versionVector );
}

/**
 * Adding version numbers to versionComboBox
 * @param versionVector : vector of version numbers
 */
public void setVersionComboBox( Vector versionVector ){
    this.versionComboBox.removeAllItems();

    for( int i=0; i < versionVector.size(); i++ ){
        String vv = (String)versionVector.elementAt( i );
        this.versionComboBox.addItem( vv.trim() );
    }
}

/**
 * Adding version numbers to currentStepComboBox and mergedStepComboBox
 * @param vc : vector of VersionControl objects
 */
public void setInitialFrame( VersionControl vc ){
    if( this.EHLComboBox.getItemCount() > 0 ){
        this.EHLComboBox.setSelectedItem(vc.getCurrentLoop());
        this.stepIDComboBox.setSelectedItem(vc.getCurrentStep());
        this.versionComboBox.setSelectedItem(vc.getCurrentVersion());
    }
}
}

```

5. Cases.GUI.AVC.AVCSplittingFrame

```

/**-----
 * Filename: AVCSplittingFrame.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1
 * Description: This class contains the AVC Splitting Step functionality.
 * Modified by removing Visual Cafe dependencies and using standard events and listeners.
 *-----
 */
package Cases.GUI.avc;

import java.awt.*;

import java.awt.event.ActionEvent;

```

```

import java.awt.event.ActionListener;

import java.awt.event.ItemEvent;

import java.awt.event.ItemListener;

import java.io.*;

import java.util.Hashtable;

import java.util.StringTokenizer;

import java.util.Vector;

import javax.swing.*;

import Cases.CasesFrame;

import Cases.CasesTitle;

import Cases.Dependency;

import Cases.EHL;

import Cases.Interfaces.I_AVC;

////////////////////////////////////
/**
 * Automatic Version Control - Evolution History Splitting : to split the existing
 * versions into new version.
 *
 * Implement CasesTitle where stores all global variables of Cases package
 * Implements interface I_AVC
 */
////////////////////////////////////
public class AVCSplittingFrame extends JFrame implements CasesTitle, I_AVC, ActionListener, ItemListener
{
    /**
     * pathName : current path name, e.g. C:\Cases\projectName, D:\Cases\projectName, ...
     */
    public String pathName = CASESDIRECTORY.getAbsolutePath();

    /** variable to link back to CaseFrame */
    private CasesFrame ownerWindow;

    /**
     * versionVector : stores all version numbers of current project
     */
    public Vector versionVector = new Vector();

    /**
     * EHLHashtable : stores all EHL objects which retrieve from loop.cfg file
     */
    public Hashtable EHLHashtable = new Hashtable();

    /**
     * depHashtable : stores all Dependency objects which retrieve from dependency.cfg file
     */
    public Hashtable depHashtable = new Hashtable();

    /**
     * Creating AVCSplittingFrame : initializes GUI and its components
     */
    public void initGUI() {
        //{{INIT_CONTROLS

```

```

setTitle("Automated Version Control - Evolution History Splitting");
getContentPane().setLayout(null);
setSize(480,280);
setVisible(false);
JLabel9.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel9.setText("Evolution Process");
getContentPane().add(JLabel9);
JLabel9.setForeground(java.awt.Color.black);
JLabel9.setBounds(15,70,145,22);
currentLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
currentLabel.setText("Current Step Version");
getContentPane().add(currentLabel);
currentLabel.setForeground(java.awt.Color.black);
currentLabel.setBounds(15,110,145,22);
OKButton.setText("OK");
OKButton.setActionCommand("jbutton");
getContentPane().add(OKButton);
OKButton.setBounds(163,210,75,22);
cancelButton.setText("Cancel");
cancelButton.setActionCommand("jbutton");
getContentPane().add(cancelButton);
cancelButton.setBounds(241,210,75,22);
newVersionTextField.setEditable(false);
getContentPane().add(newVersionTextField);
newVersionTextField.setBackground(java.awt.Color.white);
newVersionTextField.setForeground(java.awt.Color.black);
newVersionTextField.setBounds(164,150,300,22);
getContentPane().add(currentStepComboBox);
currentStepComboBox.setBounds(164,110,300,22);
JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
JLabel1.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
JLabel1.setText("Evolution History Splitting:");
getContentPane().add(JLabel1);
JLabel1.setForeground(java.awt.Color.black);
JLabel1.setFont(new Font("Dialog", Font.BOLD, 18));
JLabel1.setBounds(0,6,250,25);
projectLabel.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
projectLabel.setText("Project Label");
getContentPane().add(projectLabel);
projectLabel.setForeground(java.awt.Color.black);
projectLabel.setFont(new Font("Dialog", Font.BOLD, 18));
projectLabel.setBounds(250,6,230,25);
getContentPane().add(EHLComboBox);
EHLComboBox.setBounds(164,70,300,22);
newStepVersionButton.setText("New Step Version");
newStepVersionButton.setActionCommand("New Step Version");
getContentPane().add(newStepVersionButton);
newStepVersionButton.setBounds(25,150,135,22);
newStepVersionButton.setLabel("New Step Variant");

    //}

    //{{INIT_MENU
    //}}

    //{{REGISTER_LISTENERS

    OKButton.addActionListener(this);
    cancelButton.addActionListener(this);

    currentStepComboBox.addItemListener(this);
    EHLComboBox.addItemListener(this);
    newStepVersionButton.addActionListener(this);
    //}}
}

```

/**

* To be called when Evolution History Splitting Menu Item of CasesFrame


```

*   receives the event from user.
*   Retrieving Dependency and EHL object from dependency.cfg and loop.cfg files
*/
    public AVCSplittingFrame( CasesFrame owner ){
ownerWindow = owner;
initGUI();
        this.projectLabel.setText( ownerWindow.projectName );

try{
    FileInputStream fileInput = new FileInputStream( pathName+"\\\\"+ownerWindow.projectName+"\\dependency.cfg" );
    ObjectInputStream dep = new ObjectInputStream( fileInput );
    if( dep != null ){
        this.depHashtable = (Hashtable) dep.readObject();
    }
    dep.close();
    fileInput.close();
}
catch( IOException e ){
    debug("IOException_Dep: "+e);
}
catch( ClassNotFoundException ex ){
    debug("ClassNotFoundException_Dep: "+ex);
}
}

try{
    FileInputStream fileInput = new FileInputStream( pathName+"\\\\"+ownerWindow.projectName+"\\loop.cfg" );
    ObjectInputStream loopIn = new ObjectInputStream( fileInput );
    if( loopIn != null ){
        Vector loopVector = new Vector();
        loopVector = (Vector)loopIn.readObject();
        if( loopVector.size() > 0 ){
            this.setLoopNameComboBox( loopVector );
        }
    }
    loopIn.close();
    fileInput.close();
}
catch( IOException e ){
    debug("IOException_loop: "+e);
}
catch( ClassNotFoundException ex ){
    debug("ClassNotFoundException_loop: "+ex);
}
try { EHLComboBox.setSelectedIndex(1); }
catch (IllegalArgumentException ex) { debug("Must lock project schema!" +ex); }
}

/**
* procedure to set title on frame
* @param String
*/
    public AVCSplittingFrame(String sTitle)    {
        setTitle(sTitle);
    }

/**
* procedure to set visibility based on b
* @param boolean
*/
    public void setVisible(boolean b)
    {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

    /** overrides super method addNotify() */
    public void addNotify()    {

```

```

        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    //{{DECLARE_CONTROLS
    JLabel JLabel9 = new javax.swing.JLabel();
    JLabel currentLabel = new javax.swing.JLabel();
    JButton OKButton = new javax.swing.JButton();
    JButton cancelButton = new javax.swing.JButton();
    JTextField newVersionTextField = new javax.swing.JTextField();
    JComboBox currentStepComboBox = new javax.swing.JComboBox();
    JLabel JLabel1 = new javax.swing.JLabel();
    JLabel projectLabel = new javax.swing.JLabel();
    JComboBox EHLComboBox = new javax.swing.JComboBox();
    JButton newStepVersionButton = new javax.swing.JButton();
    //}}

    //{{DECLARE_MENUS
    //}}

    /** procedure for tracking actions performed */
    public void actionPerformed(ActionEvent event)    {
        Object object = event.getSource();
        if (object == OKButton)
            OKButton_actionPerformed(event);
        else if (object == cancelButton)
            cancelButton_actionPerformed(event);
        else if (object == newStepVersionButton)
            newStepVersionButton_actionPerformed(event);
    }

    /**
     * Create new directory with new version number for all steps of the current project
     * @param event, occur when user press OK button
     */
    public void OKButton_actionPerformed(ActionEvent event)    {
        checkPath(this.versionVector);
        setVisible( false );
        dispose();
    }

    /**
     * Exit AVCSplittingFrame
     * @param event, occur when user press Cancel button
     */
    public void cancelButton_actionPerformed(ActionEvent event)    {
        setVisible( false );
        dispose();
    }

    /**

```

```

        * Create new version number and the default version is 1.1
        * @param event, occur when user press New Version button
        */
        public void newStepVersionButton_actionPerformed(ActionEvent event)    {
            if( (EHLComboBox.getItemCount() > 0) && (EHLComboBox.getSelectedIndex() > 0) ){
                if( this.currentStepComboBox.getItemCount() == 0 ){
                    newVersionTextField.setText("1.1");
                }
                else{
                    createVersionNumber();
                }
            }
        }
        /** procedure for tracking item state changes of combo boxes */
        public void itemStateChanged(ItemEvent event)    {
            Object object = event.getSource();
            if (object == currentStepComboBox)
                currentStepComboBox_itemStateChanged(event);
            else if (object == EHLComboBox)
                EHLComboBox_itemStateChanged(event);
        }

    /**
     * Allows a user to select all the available Evolution processes in the combobox
     * @param event, occur when user select Evolution Process
     */
    public void EHLComboBox_itemStateChanged(ItemEvent event)  {
        if( event.getStateChange() == ItemEvent.SELECTED ){
            String selectedItem = (String)event.getItem();
            int selectedIndex = this.EHLComboBox.getSelectedIndex();

            if( selectedIndex > 0 ){
                if( this.EHLHashtable.containsKey( selectedItem ) ){
                    EHL ehl = (EHL)this.EHLHashtable.get( selectedItem );
                    this.versionVector = new Vector();
                    this.versionVector = tokenizeVector((String)ehl.getEHLPath());
                    this.setVersionComboBox( this.versionVector);
                }
            }
            else if( selectedIndex == 0 ){
                this.currentStepComboBox.removeAllItems();
                this.newVersionTextField.setText("");
            }
        }
    }

    /**
     * Allows a user to select all the available steps in the combobox
     * @param event, occur when user select currentStepComboBox
     */
    public void currentStepComboBox_itemStateChanged(ItemEvent event)    {
        if( event.getStateChange() == ItemEvent.SELECTED ){
            newVersionTextField.setText("");
        }
    }

    /**
     * Short cut to print the output
     * @param string : the output string
     */
    public void debug( String string ){
        System.out.println( string );
    }

    /**
     * Tokenizing a string
     * @param string : tokenized string
     * @return v : vector of string without ","

```

```

*/
    public Vector tokenizeVector(String string){
StringTokenizer st = new StringTokenizer( string, " , " );
        Vector v = new Vector();
        while( st.hasMoreTokens() ){
            v.addElement( st.nextToken() );
        }
    return v;
    }

/**
 * Adding evolution processes into EHL.ComboBox
 * @param loopVector : vector of evolution processes
 */
    public void setLoopNameComboBox( Vector loopVector ){
        EHL.ComboBox.removeAllItems();
        EHL.ComboBox.addItem(PROCESS_TITLE);

        this.EHLHashtable = new Hashtable();

        for( int i=0; i<loopVector.size(); i++ ){
            EHL ehl = (EHL)loopVector.elementAt( i );
            String loopName = ehl.getEHLName();

            //set step Hashtable
            this.EHLHashtable.put( loopName, ehl );

            EHL.ComboBox.addItem(loopName);
        }
    }

/**
 * Adding version numbers to currentStepComboBox
 * @param versionVector : vector of version numbers
 */
    public void setVersionComboBox( Vector versionVector ){
        this.currentStepComboBox.removeAllItems();
        File aFile = new File(this.pathName + "\\\" + ownerWindow.projectName, (String)versionVector.elementAt(0));
        if( aFile.isDirectory() ){
            String[] list = aFile.list();

            if( list.length > 0 ){
                for( int i=0; i<list.length; i++ ){
                    this.currentStepComboBox.addItem(((String)list[i]).trim());
                }
            }
        }
    }

/**
 * Create new version number for all steps in the current project
 * new version = highest version + 1.1
 */
    public void createVersionNumber(){
        try{
            int variantMax = findMaxVariant() + 1;
            String current = ((String)currentStepComboBox.getSelectedItem()).trim();
            int index = current.indexOf(".");
            String sub1 = current.substring(0, index);
            String sub2 = current.substring(index+1);
            int i1 = Integer.parseInt(sub1)+1;
            int i2 = Integer.parseInt(sub2)+1;
            String stepVersion = variantMax+"."+i2;
            for( int i=0; i<currentStepComboBox.getItemCount(); i++ ){
                String s = (String)currentStepComboBox.getItemAt(i);
                if( s.equals(stepVersion) ){
                    JOptionPane.showMessageDialog(this, stepVersion+" version already exists in this project!",

```

```

        "Error Message", JOptionPane.ERROR_MESSAGE);
    return;
}
else if( i==currentStepComboBox.getItemCount()-1){
    newVersionTextField.setText(stepVersion);
}
}
}
catch( Exception e){}
}

/**
 * Go to each step (e.g., s-I, s-C, s-R, ...) and create new directory with
 * the name is new version number
 * @param v : vector of step names
 */
public void checkPath(Vector v){
    String mergedVersion = this.newVersionTextField.getText();
    ownerWindow.drawPanel.addNewVersionQFDInformation(mergedVersion.trim());
    if( v.size() > 0 ){
        File myFile = new File(this.pathName + "\\\" + ownerWindow.projectName);
        if( myFile.exists() ) {
            String[] myList = myFile.list();
            for(int m=0; m<myList.length; m++){
                File f = new File(myFile, (String)myList[m]);
                if( f.isDirectory() ){
                    if( v.contains(f.getName()) ){
                        File theFile = new File(f,mergedVersion);
                        theFile.mkdir();
                        if( !theFile.isDirectory() ){
                            theFile.mkdir();
                        }
                        if( theFile.isDirectory() ){
                            Dependency dep = (Dependency) this.depHashtable.get( f.getName() );
                            createFiles(dep, theFile);
                        }
                    }
                }
            }
        }
    }
}

/**
 * Create Component Content directory and link files inside it.
 * @param dep : Dependency object of theFile and get input.p and input.s files
 * @param theFile : current version directory
 */
public void createFiles( Dependency dep, File theFile){
    if( dep != null ){
        try{
            //Create Component Content subdirectory in thte new step
            //and include txt.link, data.link, url.link, and caps.link
            File compContent = new File(theFile,"Component Content");
            compContent.mkdir();
            if( compContent.isDirectory() ){
                for( int i=0; i<LINK_FILE_NAMES.length; i++ ){
                    File newFile = new File(compContent, LINK_FILE_NAMES[i]);
                    FileWriter fileWriter = new FileWriter( newFile );
                    BufferedWriter bw = new BufferedWriter( fileWriter );
                    bw.flush();
                    bw.close();
                    fileWriter.close();
                }
            }
        }
    }

    FileOutputStream fileOutput = new FileOutputStream(theFile.getAbsolutePath()+"\\input.p");
    DataOutputStream depPrimary = new DataOutputStream( fileOutput);

```

```

        if( depPrimary != null ){
            depPrimary.writeBytes(dep.getPrimaryInput());
        }
        depPrimary.flush();
        depPrimary.close();
        fileOutput.close();

        fileOutput = new FileOutputStream(theFile.getAbsolutePath()+"\\input.s");
        DataOutputStream depSecondary = new DataOutputStream( fileOutput);
        if( depSecondary != null ){
            depSecondary.writeBytes(dep.getSecondaryInput());
        }
        depSecondary.flush();
        depSecondary.close();
        fileOutput.close();
    }
    catch( IOException e ){
        debug("dep_IOException: "+e);
    }
}
}
/**
 * Find the maximum variant number of the step
 * @return variantMax : the maximum variant number of this process
 */
public int findMaxVariant(){
    int variantMax = 0;
    File aFile = new File(this.pathName + "\\\" + ownerWindow.projectName, (String)versionVector.elementAt(0));
    if( aFile.isDirectory() ){
        String[] list = aFile.list();

        if( list.length > 0 ){
            Vector v = new Vector();
            for( int i=0; i<list.length; i++ ){
                String s = list[i];
                int index = s.indexOf(".");
                String sub1 = (s.substring(0,index)).trim();
                if( !v.contains(sub1) ){
                    v.addElement(sub1);
                }
            }
            if( v.size() > 0 ){
                try{
                    variantMax = Integer.parseInt((String)v.elementAt(0));
                    for(int j=1; j<v.size(); j++){
                        variantMax = Math.max(variantMax,Integer.parseInt((String)v.elementAt(j)));
                    }
                }
                catch(Exception e){}
            }
        }
    }
    return variantMax;
}
}

```

D. CASES.GUI.CALENDARDIALOG PACKAGE

1. Cases.GUI.CalendarDialog.DateButton

```
/**-----
 * Filename: DateButton.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1
 * Description: This class provides a date button.
 * Modified by removing Visual Cafe dependencies and using standard events and listeners.
 *-----
 */
package Cases.GUI.CalendarDialog;

import java.awt.event.ActionEvent;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.swing.*;

/**
 * Custom button for entering dates. The <code>DateButton</code> class
 * is just a standard button that defines an additional bound
 * property: "date". The button displays the date property as its
 * label. When clicked, it does not generate an
 * <code>ActionEvent</code>, but displays a {@link DateChooser} dialog
 * instead, that allows you to change the date. When the date is
 * changed, a <code>PropertyChangeEvent</code> is generated, according
 * the contract for bound properties.
 */
public class DateButton extends JButton
{
    /** Format to use to display the date property. */
    private static final DateFormat DATE_FORMAT = new SimpleDateFormat("MM-dd-yyyy");

    /** DateChooser instance to use to change the date. */
    private static final DateChooser DATE_CHOOSER = new DateChooser((JFrame)null,"Select Date");

    /** Date property. */
    private Date date;

    /**
     * Called when the button is clicked, in order to fire an
     * <code>ActionEvent</code>. Displays the dialog to change the
     * date instead of generating the event and updates the date
     * property.
     *
     * @param e <code>ActionEvent</code> to fire
     */
    protected void fireActionPerformed( ActionEvent e ) {
        Date newDate = DATE_CHOOSER.select(date);
        if ( newDate == null )
            return;
        setDate( newDate );
    }

    /**
     * Constructs a new <code>DateButton</code> object with a given
     * date.
     *
     * @param date initial date
     */
}
```

```

public DateButton( Date date ) {
    super( DATE_FORMAT.format(date) );
    this.date = date;
}

/**
 * Constructs a new <code>DateButton</code> object with the system
 * date as the initial date.
 */
public DateButton() {
    this( new Date() );
}

/**
 * Gets the value of the date property.
 *
 * @return the current value of the date property
 */
public Date getDate() {
    return date;
}

/**
 * Sets the value of the date property.
 *
 * @param date new value of the date property
 *
 * @return the old value of the date property
 */
public void setDate( Date date ) {
    Date old = this.date;
    this.date = date;
    setText( DATE_FORMAT.format(date) );
    firePropertyChange( "date", old, date );
}
}

```

2. Cases.GUI.CalendarDialog.DateChooser

```

/**-----
 * Filename: DateChooser.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1
 * Description: This class provides a calendar dialog for assigning dates to tasks.
 * Modified by removing Visual Cafe dependencies and using standard events and listeners.
 *-----
 */
package Cases.GUI.CalendarDialog;

import java.awt.*;
import java.awt.event.*;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import javax.swing.*;
import javax.swing.border.Border;

/**
 * Custom dialog box to enter dates. The <code>DateChooser</code>
 * class presents a calendar and allows the user to visually select a
 * day, month and year so that it is impossible to enter an invalid
 * date.
 */
public class DateChooser extends JDialog

```



```

implements ItemListener, MouseListener, FocusListener, KeyListener, ActionListener
{
    /** Names of the months. */
    private static final String[] MONTHS =
        new String[] {
            "January",      "February",      "March",
            "April",      "May",      "June",
            "July",      "August",      "September",
            "October",      "November",      "December"
        };

    /** Names of the days of the week. */
    private static final String[] DAYS =
        new String[] {
            "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
        };

    /** Text color of the days of the weeks, used as column headers in
        the calendar. */
    private static final Color WEEK_DAYS_FOREGROUND = Color.black;

    /** Text color of the days' numbers in the calendar. */
    private static final Color DAYS_FOREGROUND = Color.blue;

    /** Background color of the selected day in the calendar. */
    private static final Color SELECTED_DAY_FOREGROUND = Color.white;

    /** Text color of the selected day in the calendar. */
    private static final Color SELECTED_DAY_BACKGROUND = Color.blue;

    /** Empty border, used when the calendar does not have the focus. */
    private static final Border EMPTY_BORDER = BorderFactory.createEmptyBorder(1,1,1,1);

    /** Border used to highlight the selected day when the calendar
        has the focus. */
    private static final Border FOCUSED_BORDER = BorderFactory.createLineBorder(Color.yellow,1);

    /** First year that can be selected. */
    private static final int FIRST_YEAR = 1900;

    /** Last year that can be selected. */
    private static final int LAST_YEAR = 2100;

    /** Auxiliary variable to compute dates. */
    private GregorianCalendar calendar;

    /** Calendar, as a matrix of labels. The first row represents the
        first week of the month, the second row, the second week, and
        so on. Each column represents a day of the week, the first is
        Sunday, and the last is Saturday. The label's text is the
        number of the corresponding day. */
    private JLabel[][] days;

    /** Day selection control. It is just a panel that can receive the
        focus. The actual user interaction is driven by the
        <code>DateChooser</code> class. */
    private FocusablePanel daysGrid;

    /** Month selection control. */
    private JComboBox month;

    /** Year selection control. */
    private JComboBox year;

    /** "Ok" button. */
    private JButton ok;

    /** "Cancel" button. */

```

```

private JButton cancel;

/** Day of the week (0=Sunday) corresponding to the first day of
    the selected month. Used to calculate the position, in the
    calendar ({@link #days}), corresponding to a given day. */
private int offset;

/** Last day of the selected month. */
private int lastDay;

/** Selected day. */
private JLabel day;

/** <code>true</code> if the "Ok" button was clicked to close the
    dialog box, <code>false</code> otherwise. */
private boolean okClicked;

/**
 * Custom panel that can receive the focus. Used to implement the
 * calendar control.
 */
private static class FocusablePanel extends JPanel
{
    /**
     * Constructs a new <code>FocusablePanel</code> with the given
     * layout manager.
     *
     * @param layout layout manager
     */
    public FocusablePanel( LayoutManager layout ) {        super( layout );    }

    /**
     * Always returns <code>true</code>, since
     * <code>FocusablePanel</code> can receive the focus.
     *
     * @return <code>true</code>
     */
    public boolean isFocusTraversable() {        return true;    }
}

/**
 * Initializes this <code>DateChooser</code> object. Creates the
 * controls, registers listeners and initializes the dialog box.
 */
private void construct() {
    calendar = new GregorianCalendar();

    month = new JComboBox(MONTHS);
    month.addItemListener( this );

    year = new JComboBox();
    for ( int i=FIRST_YEAR; i<=LAST_YEAR; i++ )
        year.addItem( Integer.toString(i) );
    year.addItemListener( this );

    days = new JLabel[7][7];
    for ( int i=0; i<7; i++ ) {
        days[0][i] = new JLabel(DAYS[i],JLabel.RIGHT);
        days[0][i].setForeground( WEEK_DAYS_FOREGROUND );
    }
    for ( int i=1; i<7; i++ )
        for ( int j=0; j<7; j++ )
        {
            days[i][j] = new JLabel(" ",JLabel.RIGHT);
            days[i][j].setForeground( DAYS_FOREGROUND );
            days[i][j].setBackground( SELECTED_DAY_BACKGROUND );
            days[i][j].setBorder( EMPTY_BORDER );
            days[i][j].addMouseListener( this );
        }
}

```

```

        }

        ok = new JButton("Ok");
        ok.addActionListener( this );
        cancel = new JButton("Cancel");
        cancel.addActionListener( this );

        JPanel monthYear = new JPanel();
        monthYear.add( month );
        monthYear.add( year );

        daysGrid = new FocusablePanel(new GridLayout(7,7,5,0));
        daysGrid.addFocusListener( this );
        daysGrid.addKeyListener( this );
        for ( int i=0; i<7; i++ )
            for ( int j=0; j<7; j++ )
                daysGrid.add( days[i][j] );
        daysGrid.setBackground( Color.white );
        daysGrid.setBorder( BorderFactory.createLoweredBevelBorder() );
        JPanel daysPanel = new JPanel();
        daysPanel.add( daysGrid );

        JPanel buttons = new JPanel();
        buttons.add( ok );
        buttons.add( cancel );

        Container dialog = getContentPane();
        dialog.add( "North", monthYear );
        dialog.add( "Center", daysPanel );
        dialog.add( "South", buttons );

        pack();
        setResizable( false );
    }

    /**
     * Gets the selected day, as an <code>int</code>. Parses the text
     * of the selected label in the calendar to get the day.
     *
     * @return the selected day or -1 if there is no day selected
     */
    private int getSelectedDay() {
        if ( day == null )
            return -1 ;
        try {
            return Integer.parseInt(day.getText());
        } catch ( NumberFormatException e ) {
        }
        return -1;
    }

    /**
     * Sets the selected day. The day is specified as the label
     * control, in the calendar, corresponding to the day to select.
     *
     * @param newDay day to select
     */
    private void setSelected( JLabel newDay ) {
        if ( day != null ) {
            day.setForeground( DAYS_FOREGROUND );
            day.setOpaque( false );
            day.setBorder( EMPTY_BORDER );
        }
        day = newDay;
        day.setForeground( SELECTED_DAY_FOREGROUND );
        day.setOpaque( true );
        if ( daysGrid.hasFocus() )
            day.setBorder( FOCUSED_BORDER );
    }

```

```

}

/**
 * Sets the selected day. The day is specified as the number of
 * the day, in the month, to selected. The function compute the
 * corresponding control to select.
 *
 * @param newDay day to select
 */
private void setSelected( int newDay ) {
    setSelected( days[(newDay+offset-1)/7+1][(newDay+offset-1)%7] );
}

/**
 * Updates the calendar. This function updates the calendar panel
 * to reflect the month and year selected. It keeps the same day
 * of the month that was selected, except if it is beyond the last
 * day of the month. In this case, the last day of the month is
 * selected.
 */
private void update() {
    int iday = getSelectedDay();
    for ( int i=0; i<7; i++ ) {
        days[1][i].setText( " " );
        days[5][i].setText( " " );
        days[6][i].setText( " " );
    }
    this.calendar.set( Calendar.DATE, 1 );
    this.calendar.set( Calendar.MONTH, month.getSelectedIndex()+Calendar.JANUARY );
    this.calendar.set( Calendar.YEAR, year.getSelectedIndex()+FIRST_YEAR );

    offset = calendar.get(Calendar.DAY_OF_WEEK)-Calendar.SUNDAY;
    lastDay = calendar.getActualMaximum(Calendar.DATE);
    for ( int i=0; i<lastDay; i++ )
        days[(i+offset)/7+1][(i+offset)%7].setText( String.valueOf(i+1) );
    if ( iday != -1 ) {
        if ( iday > lastDay )
            iday = lastDay;
        setSelected( iday );
    }
}

/**
 * Called when the "Ok" button is pressed. Just sets a flag and
 * hides the dialog box.
 */
public void actionPerformed( ActionEvent e ) {
    if ( e.getSource() == ok )
        okClicked = true;
    hide();
}

/**
 * Called when the calendar gains the focus. Just re-sets the
 * selected day so that it is redrawn with the border that
 * indicate focus.
 */
public void focusGained( FocusEvent e ) {
    setSelected( day );
}

/**
 * Called when the calendar loses the focus. Just re-sets the
 * selected day so that it is redrawn without the border that
 * indicate focus.
 */
public void focusLost( FocusEvent e ) {
    setSelected( day );
}

```

```

    }

    /**
     * Called when a new month or year is selected. Updates the calendar
     * to reflect the selection.
     */
    public void itemStateChanged( ItemEvent e ) {
        update();
    }

    /**
     * Called when a key is pressed and the calendar has the
     * focus. Handles the arrow keys so that the user can select a day
     * using the keyboard.
     */
    public void keyPressed( KeyEvent e ) {
        int iday = getSelectedDay();
        switch ( e.getKeyCode() ) {
            case KeyEvent.VK_LEFT:
                if ( iday > 1 )
                    setSelected( iday-1 );
                break;
            case KeyEvent.VK_RIGHT:
                if ( iday < lastDay )
                    setSelected( iday+1 );
                break;
            case KeyEvent.VK_UP:
                if ( iday > 7 )
                    setSelected( iday-7 );
                break;
            case KeyEvent.VK_DOWN:
                if ( iday <= lastDay-7 )
                    setSelected( iday+7 );
                break;
        }
    }

    /**
     * Called when the mouse is clicked on a day in the
     * calendar. Selects the clicked day.
     */
    public void mouseClicked( MouseEvent e ) {
        JLabel day = (JLabel)e.getSource();
        if ( !day.getText().equals(" ") )
            setSelected( day );
        daysGrid.requestFocus();
    }

    // standard key events for future enhancements
    public void keyReleased( KeyEvent e ) {}
    public void keyTyped( KeyEvent e ) {}

    // standard mouse events for future enhancements
    public void mouseEntered( MouseEvent e ) {}
    public void mouseExited( MouseEvent e ) {}
    public void mousePressed( MouseEvent e ) {}
    public void mouseReleased( MouseEvent e ) {}

    /**
     * Constructs a new <code>DateChooser</code> with the given title.
     *
     * @param owner owner dialog
     *
     * @param title dialog title
     */
    public DateChooser( Dialog owner, String title ) {
        super( owner, title, true );
        construct();
    }

```

```

    }

    /**
     * Constructs a new <code>DateChooser</code>.
     *
     * @param owner owner dialog
     */
    public DateChooser( Dialog owner ) {
        super( owner, true );
        construct();
    }

    /**
     * Constructs a new <code>DateChooser</code> with the given title.
     *
     * @param owner owner frame
     *
     * @param title dialog title
     */
    public DateChooser( Frame owner, String title ) {
        super( owner, title, true );
        construct();
    }

    /**
     * Constructs a new <code>DateChooser</code>.
     *
     * @param owner owner frame
     */
    public DateChooser( Frame owner ) {
        super( owner, true );
        construct();
    }

    /**
     * Selects a date. Displays the dialog box, with a given date as
     * the selected date, and allows the user select a new date.
     *
     * @param date initial date
     *
     * @return the new date selected or <code>null</code> if the user
     * press "Cancel" or closes the dialog box
     */
    public Date select( Date date ) {
        calendar.setTime( date );
        int _day = calendar.get( Calendar.DATE );
        int _month = calendar.get( Calendar.MONTH );
        int _year = calendar.get( Calendar.YEAR );

        year.setSelectedIndex( _year-FIRST_YEAR );
        month.setSelectedIndex( _month-Calendar.JANUARY );
        setSelected( _day );
        okClicked = false;
        show();
        if ( !okClicked )
            return null;
        calendar.set( Calendar.DATE, getSelectedDay() );
        calendar.set( Calendar.MONTH, month.getSelectedIndex()+Calendar.JANUARY );
        calendar.set( Calendar.YEAR, year.getSelectedIndex()+FIRST_YEAR );
        return calendar.getTime();
    }

    /**
     * Selects new date. Just calls {@link #select(Date)} with the
     * system date as the parameter.
     *
     * @return the same as the function {@link #select(Date)}
     */

```

```

    public Date select() {        return select(new Date());    }
}

```

E. CASES.GUI.PSF PACKAGE

1. Cases.GUI.PSF.ComponentTypePanel

```

/**-----
 * @Filename: ComponentTypePanel.java
 * @Date: 3-7-2003
 * @Author: Hahn Le
 * Modified by Arthur Clomera for NPS Thesis
 * @Compiler: JDK 1.3.1
 * Modified by removing Visual Cafe dependencies and using standard events and listeners.
 * Removing from the PSF to make it a coherent object and adding QFD functionality
 **/

package Cases.GUI.psf;

import java.awt.*;
import javax.swing.*;
import Cases.ProjectSchemaFrame;

/** this class is the panel for editing, deleting, and adding component types */
public class ComponentTypePanel extends JPanel {

    JLabel label = new JLabel();
    // label to display project name
    public JLabel projectLabel = new JLabel();
    JLabel JLabel6 = new JLabel();
    JLabel JLabel7 = new JLabel();
    JLabel JLabel17 = new JLabel();
    public JLabel compLabel = new JLabel();
    // delete, clear, edit, and save button
    public JButton compDeleteButton = new JButton();
    public JButton compClearButton = new JButton();
    public JButton compEditButton = new JButton();
    public JButton compSaveButton = new JButton();
    // text fields for component ID and Name
    public JTextField compIDTextField = new JTextField();
    public JTextField compNameTextField = new JTextField();
    // text area for component description
    public JTextArea compDescriptionTextArea = new JTextArea();

    /**
     * main constructor to initialize GUI and add action listeners from PSF
     * @param ProjectSchemaFrame
     */
    public ComponentTypePanel(ProjectSchemaFrame PSF) {
        initGUI();
        compEditButton.addActionListener(PSF);
        compDeleteButton.addActionListener(PSF);
        compClearButton.addActionListener(PSF);
        compSaveButton.addActionListener(PSF);
    }
    /**
     * procedure for initializing GUI and its components.
     */
    private void initGUI() {
        this.setLayout(null);
        this.setBackground(new java.awt.Color(204,204,204));
        this.setBounds(2,27,575,330);
        this.setVisible(false);
    }
}

```

```

JLabel17.setHorizontalAlignment( SwingConstants.RIGHT);
JLabel17.setText("Project Label:");
this.add(JLabel17);
JLabel17.setForeground(java.awt.Color.black);
JLabel17.setFont(new Font("Dialog", Font.BOLD, 18));
JLabel17.setBounds(0,6,280,24);
compLabel.setText("Label");
this.add(compLabel);
compLabel.setForeground(java.awt.Color.black);
compLabel.setFont(new Font("Dialog", Font.BOLD, 18));
compLabel.setBounds(294,6,280,24);
JLabel6.setHorizontalAlignment( SwingConstants.RIGHT);
JLabel6.setText("Component Type Name");
this.add(JLabel6);
JLabel6.setForeground(java.awt.Color.black);
JLabel6.setBounds(75,110,144,22);
JLabel7.setHorizontalAlignment( SwingConstants.RIGHT);
JLabel7.setText("Component Type Description");
this.add(JLabel7);
JLabel7.setForeground(java.awt.Color.black);
JLabel7.setBounds(new java.awt.Rectangle(52, 144, 170, 22));

compDeleteButton.setText("Delete");
compDeleteButton.setActionCommand("jbutton");

compDeleteButton.setBounds(168,297,75,22);
compClearButton.setText("Clear");
compClearButton.setActionCommand("jbutton");

compClearButton.setBounds(246,297,75,22);
compEditButton.setText("Edit");
compEditButton.setActionCommand("jbutton");
this.add(compEditButton);
compEditButton.setBounds(90,297,75,22);

this.add(compNameTextField);
compNameTextField.setBounds(225,110,300,22);

compDescriptionTextArea.setLineWrap(true);
compDescriptionTextArea.setWrapStyleWord(true);
this.add(compDescriptionTextArea);
compDescriptionTextArea.setBounds(new java.awt.Rectangle(225, 143, 300, 80));
compSaveButton.setText("Save");
compSaveButton.setActionCommand("Save");

compSaveButton.setBounds(492,297,75,22);
}
}

```

2. Cases.GUI.PSF.DependencyTypePanel

```

/**-----
 * @Filename: DependencyTypePanel.java
 * @Date: 3-7-2003
 * @Author: Arthur Clomera for NPS Thesis
 * @Compiler: JDK 1.3.1
 * Description: this class is the panel for editing, deleting, and adding Dependency types
 * Modified by removing Visual Cafe dependencies and using standard events and listeners.
 * Removing from the PSF to make it a coherent object and adding QFD functionality
 * -----
 */

package Cases.GUI.psf;

import java.awt.*;

```



```

import javax.swing.*;
import Cases.ProjectSchemaFrame;
/** this class is the panel for editing, deleting, and adding Dependency types */
public class DependencyTypePanel extends JPanel {
    JLabel JLabel9 = new JLabel();
    JLabel JLabel10 = new JLabel();
    JLabel JLabel12 = new JLabel();

    /** okay button */
    public JButton depenOKButton = new JButton();
    /** cancel button */
    public JButton depenCancelButton = new JButton();
    /** secondary input component types text field */
    public JTextField depenSecondaryTextField = new JTextField();
    /** output component type text field */
    public JTextField depenOutputTextField = new JTextField();
    /** combo box of available steps */
    public JComboBox depenStepComboBox = new JComboBox();
    /** primary input component type text field */
    public JTextField depenPrimaryTextField = new JTextField();
    JLabel JLabel18 = new JLabel();
    public JLabel depLabel = new JLabel();
    JLabel JLabel19 = new JLabel();
    /** combo box of evolution processes (abc) */
    public JComboBox depenEHLComboBox = new JComboBox();
    /** secondary input component button */
    public JButton secondaryButton = new JButton();
    /** update files with changes button */
    public JButton depenUpdateButton = new JButton();
    /**
     * constructor to initialize gui and add action listeners from PSF
     * @param ProjectSchemaFrame
     */
    public DependencyTypePanel(ProjectSchemaFrame psf) {
        initGUI();
        depenOKButton.addActionListener(psf);
        depenCancelButton.addActionListener(psf);
        depenStepComboBox.addItemListener(psf);
        depenEHLComboBox.addItemListener(psf);
        secondaryButton.addActionListener(psf);
        depenUpdateButton.addActionListener(psf);
    }

    /**
     * procedure to initialize GUI and its components
     */
    private void initGUI() {
        this.setLayout(null);
        this.setBounds(2,27,575,330);
        this.setVisible(false);
        JLabel9.setHorizontalAlignment( SwingConstants.RIGHT);
        JLabel9.setText("Step Types");
        this.add(JLabel9);
        JLabel9.setForeground(java.awt.Color.black);
        JLabel9.setBounds(30,110,240,22);
        JLabel10.setHorizontalAlignment( SwingConstants.RIGHT);
        JLabel10.setText("Output Component Type");
        this.add(JLabel10);
        JLabel10.setForeground(java.awt.Color.black);
        JLabel10.setBounds(30,150,240,22);
        JLabel12.setHorizontalAlignment( SwingConstants.RIGHT);
        JLabel12.setText("Primary Input Component Type");
        this.add(JLabel12);
        JLabel12.setForeground(java.awt.Color.black);
        JLabel12.setBounds(30,190,240,22);
        depenOKButton.setText("OK");
        depenOKButton.setActionCommand("jbutton");
        this.add(depenOKButton);
        depenOKButton.setBounds(390,290,75,22);
        depenCancelButton.setText("Cancel");

```

```

        depenCancelButton.setActionCommand("jbutton");
        this.add(depenCancelButton);
        depenCancelButton.setBounds(470,290,75,22);
        depenSecondaryTextField.setEditable(false);
        this.add(depenSecondaryTextField);
        depenSecondaryTextField.setBackground(java.awt.Color.white);
        depenSecondaryTextField.setBounds(275,230,270,22);
        depenOutputTextField.setEditable(false);
        this.add(depenOutputTextField);
        depenOutputTextField.setBackground(java.awt.Color.white);
        depenOutputTextField.setForeground(java.awt.Color.black);
        depenOutputTextField.setFont(new Font("SansSerif", Font.PLAIN, 12));
        depenOutputTextField.setBounds(275,150,270,22);
        this.add(depenStepComboBox);
        depenStepComboBox.setBounds(275,110,270,22);
        depenPrimaryTextField.setEditable(false);
        this.add(depenPrimaryTextField);
        depenPrimaryTextField.setBackground(java.awt.Color.white);
        depenPrimaryTextField.setForeground(java.awt.Color.black);
        depenPrimaryTextField.setBounds(275,190,270,22);
        JLabel18.setHorizontalAlignment( SwingConstants.RIGHT);
        JLabel18.setText("Project Label.");
        this.add(JLabel18);
        JLabel18.setForeground(java.awt.Color.black);
        JLabel18.setFont(new Font("Dialog", Font.BOLD, 18));
        JLabel18.setBounds(0,6,280,24);
        depLabel.setText("Label");
        this.add(depLabel);
        depLabel.setForeground(java.awt.Color.black);
        depLabel.setFont(new Font("Dialog", Font.BOLD, 18));
        depLabel.setBounds(294,6,280,24);
        JLabel19.setHorizontalAlignment( SwingConstants.CENTER);
        JLabel19.setHorizontalAlignment( SwingConstants.RIGHT);
        JLabel19.setText("Evolution Process");
        this.add(JLabel19);
        JLabel19.setForeground(java.awt.Color.black);
        JLabel19.setBounds(30,70,240,22);
        this.add(depenEHLComboBox);
        depenEHLComboBox.setBounds(275,70,270,22);
        secondaryButton.setText("Secondary Input Component Type(s)");
        secondaryButton.setActionCommand("Secondary Input Component Type(s)");
        this.add(secondaryButton);
        add(depenUpdateButton);
        secondaryButton.setBounds(30,230,240,22);
        depenUpdateButton.setText("jButton1");
        depenUpdateButton.setBounds(new java.awt.Rectangle(310, 289, 73, 24));
        depenUpdateButton.setActionCommand("updateButton");
        depenUpdateButton.setLabel("Update");
    }
}

```

3. Cases.GUI.PSF.EHLTypePanel

```

/**-----
 * @Filename: EHLTypePanel.java
 * @Date: 3-7-2003
 * @Author: Arthur Clomera for NPS Thesis
 * @Compiler: JDK 1.3.1
 * Description: this class is the panel for editing, deleting, and adding EHL types
 * Modified by removing Visual Cafe dependencies and using standard events and listeners.
 * Removing from the PSF to make it a coherent object and adding QFD functionality
 * -----
 */

```

```

package Cases.GUI.psf;

```

```

import java.awt.*;
import javax.swing.*;
import Cases.ProjectSchemaFrame;
/** this class is the panel for editing, deleting, and adding EHL types */
public class EHLLTypePanel extends JPanel {
    JLabel JLabel13 = new JLabel();
    JLabel JLabel14 = new JLabel();
    /** add button for new evolution process */
    public JButton EHLLAddButton = new JButton();
    /** delete button to remove an evolution process */
    public JButton EHLLDeleteButton = new JButton();
    /** clear button to clear fields in GUI */
    public JButton EHLLClearButton = new JButton();
    /** an edit button to edit an existing evolution process */
    public JButton EHLLEditButton = new JButton();
    /** text field for Evolution Process name */
    public JTextField EHLLNameTextField = new JTextField();
    /** the evolution process path from one step to the next */
    public JTextField EHLLPathTextField = new JTextField();
    /** done button to close this frame */
    public JButton EHLLDoneButton = new JButton();
    JLabel label = new JLabel();
    public JLabel projectLabel = new JLabel();
    JLabel JLabel15 = new JLabel();
    /** drop down combo box of existing evolution processes */
    public JComboBox existedEHLLComboBox = new JComboBox();
    JLabel JLabel20 = new JLabel();
    /** scroll pane for available step types */
    public JScrollPane stepTypesScrollPane = new JScrollPane();
    /** list of available step types */
    public JList stepTypesList = new JList();
    /** update button for capturing changes */
    public JButton updateButton = new JButton();

    /**
     * this constructor initializes the gui and adds action listeners from PSF.
     * @param ProjectSchemaFrame and DefaultListModel
     */
    public EHLLTypePanel(ProjectSchemaFrame psf, DefaultListModel psfListModel) {
        initGUI();
        EHLLAddButton.addActionListener(psf);
        EHLLEditButton.addActionListener(psf);
        EHLLDeleteButton.addActionListener(psf);
        EHLLClearButton.addActionListener(psf);
        EHLLDoneButton.addActionListener(psf);
        updateButton.addActionListener(psf);
        existedEHLLComboBox.addItemListener(psf);
        stepTypesList.addMouseListener(psf);
        stepTypesList.setModel(psfListModel);
    }
    /**
     * the main procedure for initializing the GUI and its components
     */
    private void initGUI() {
        this.setDoubleBuffered(false);
        this.setLayout(null);
        this.setBounds(2,27,575,330);
        this.setVisible(false);
        JLabel13.setHorizontalAlignment( SwingConstants.RIGHT);
        JLabel13.setText("Evolution Process Name");
        this.add(JLabel13);
        JLabel13.setForeground(java.awt.Color.black);
        JLabel13.setBounds(60,70,150,22);
        JLabel14.setHorizontalAlignment( SwingConstants.RIGHT);
        JLabel14.setText("Evolution Process");
        this.add(JLabel14);
        JLabel14.setForeground(java.awt.Color.black);

```

```

JLabel14.setBounds(60,210,150,22);
EHLAddButton.setText("Add");
EHLAddButton.setActionCommand("jbutton");
this.add(EHLAddButton);
EHLAddButton.setBounds(10,297,75,22);
EHLDeleteButton.setText("Delete");
EHLDeleteButton.setActionCommand("jbutton");
this.add(EHLDeleteButton);
EHLDeleteButton.setBounds(166,297,75,22);
EHLClearButton.setText("Clear");
EHLClearButton.setActionCommand("jbutton");
this.add(EHLClearButton);
EHLClearButton.setBounds(244,297,75,22);
EHLEditButton.setText("Edit");
EHLEditButton.setActionCommand("jbutton");
this.add(EHLEditButton);
EHLEditButton.setBounds(88,297,75,22);
this.add(EHLNameTextField);
EHLNameTextField.setBounds(215,70,300,22);
EHLNameTextField.setText("abc");
EHLPathTextField.setDoubleBuffered(true);
EHLPathTextField.setEditable(false);
this.add(EHLPathTextField);
EHLPathTextField.setBackground(java.awt.Color.white);
EHLPathTextField.setForeground(java.awt.Color.black);
EHLPathTextField.setBounds(215,210,300,22);
EHLDoneButton.setText("Done");
EHLDoneButton.setActionCommand("Done");
this.add(EHLDoneButton);
EHLDoneButton.setBounds(493,297,75,22);
label.setHorizontalAlignment( SwingConstants.RIGHT);
label.setText("Project Label:");
this.add(label);
label.setForeground(java.awt.Color.black);
label.setFont(new Font("Dialog", Font.BOLD, 18));
label.setBounds(2,6,280,24);
projectLabel.setText("Label");
this.add(projectLabel);
projectLabel.setForeground(java.awt.Color.black);
projectLabel.setFont(new Font("Dialog", Font.BOLD, 18));
projectLabel.setBounds(294,6,280,24);
JLabel15.setHorizontalAlignment( SwingConstants.RIGHT);
JLabel15.setText("Existing Evolution Process");
this.add(JLabel15);
JLabel15.setForeground(java.awt.Color.black);
JLabel15.setBounds(60,250,150,22);
this.add(existedEHLComboBox);
existedEHLComboBox.setBounds(215,250,300,22);
JLabel20.setHorizontalAlignment( SwingConstants.RIGHT);
JLabel20.setText("Step Types");
this.add(JLabel20);
JLabel20.setForeground(java.awt.Color.black);
JLabel20.setBounds(60,110,150,22);
stepTypesScrollPane.setOpaque(true);
this.add(stepTypesScrollPane);
add(updateButton);
stepTypesScrollPane.setBounds(215,110,300,80);
stepTypesScrollPane.getViewport().add(stepTypesList);
stepTypesList.setBounds(0,0,297,77);
updateButton.setText("jButton1");
updateButton.setBounds(new java.awt.Rectangle(322, 297, 73, 23));
updateButton.setActionCommand("updateButton");
updateButton.setLabel("Update");
}
}

```

4. Cases.GUI.PSF.StepTypePanel

```
/**-----
 * @Filename: StepTypePanel.java
 * @Date: 3-7-2003
 * @Author: Arthur Clomera for NPS Thesis
 * @Compiler: JDK 1.3.1
 * Description: this class is the panel for editing, deleting, and adding step types
 * Modified by removing Visual Cafe dependencies and using standard events and listeners.
 * Removing from the PSF to make it a coherent object and adding QFD functionality
 *-----
 */

package Cases.GUI.psf;

import java.awt.*;
import javax.swing.*;
import Cases.ProjectSchemaFrame;

/** this class is the panel for editing, deleting, and adding step types */
public class StepTypePanel extends JPanel {

    JLabel JLabel2 = new JLabel();
    JLabel JLabel3 = new JLabel();
    /** a button to delete existing steps */
    public JButton stepDeleteButton = new JButton();
    /** a button to clear GUI fields */
    public JButton stepClearButton = new JButton();
    /** a button to edit an existing step */
    public JButton stepEditButton = new JButton();
    /** a text field for the stepID */
    public JTextField stepIDTextField = new JTextField();
    /** a text field for step name */
    public JTextField stepNameTextField = new JTextField();
    /** a text area for the step description */
    public JTextArea stepDescriptionTextArea = new JTextArea();
    /** a button to save the step information */
    public JButton stepSaveButton = new JButton();
    JLabel JLabel16 = new JLabel();
    public JLabel stepLabel = new JLabel();

    /**
     * the constructor which initializes the GUI and
     * adds action listeners from the PSF
     * @param ProjectSchemaFrame
     */
    public StepTypePanel(ProjectSchemaFrame psf) {
        initGUI();
        stepEditButton.addActionListener(psf);
        stepDeleteButton.addActionListener(psf);
        stepClearButton.addActionListener(psf);
        stepSaveButton.addActionListener(psf);
    }

    /**
     * procedure to initialize the GUI and its components
     */
    private void initGUI() {
        this.setLayout(null);
        this.setBounds(2,27,575,330);
        JLabel2.setHorizontalAlignment( SwingConstants.RIGHT);
        JLabel2.setText("Step Type Name");
        this.add(JLabel2);
        JLabel2.setForeground(java.awt.Color.black);
        JLabel2.setBounds(62,110,130,22);
        JLabel3.setHorizontalAlignment( SwingConstants.RIGHT);
        JLabel3.setText("Step Type Description");
        this.add(JLabel3);
    }
}
```

```

JLabel3.setForeground(java.awt.Color.black);
JLabel3.setBounds(new java.awt.Rectangle(73, 148, 130, 22));

// delete button
stepDeleteButton.setText("Delete");
stepDeleteButton.setActionCommand("jbutton");
stepDeleteButton.setBounds(168,297,75,22);

// clear button
stepClearButton.setText("Clear");
stepClearButton.setActionCommand("jbutton");
stepClearButton.setBounds(246,297,75,22);

// edit button
stepEditButton.setText("Edit");
stepEditButton.setActionCommand("jbutton");
stepEditButton.setBounds(90,297,75,22);
this.add(stepEditButton);

// step id
stepIDTextField.setBounds(225,70,300,22);

// step name
stepNameTextField.setBounds(225,110,300,22);
this.add(stepNameTextField);

// step description
stepDescriptionTextArea.setLineWrap(true);
stepDescriptionTextArea.setWrapStyleWord(true);
stepDescriptionTextArea.setBounds(new java.awt.Rectangle(225, 143, 300, 80));
this.add(stepDescriptionTextArea);

// save button
stepSaveButton.setText("Save");
stepSaveButton.setActionCommand("Save");
stepSaveButton.setBounds(492,297,75,22);

JLabel16.setHorizontalAlignment( SwingConstants.RIGHT);
JLabel16.setText("Project Label:");
this.add(JLabel16);
JLabel16.setForeground(java.awt.Color.black);
JLabel16.setFont(new Font("Dialog", Font.BOLD, 18));
JLabel16.setBounds(2,6,280,24);
stepLabel.setText("Label");
this.add(stepLabel);
stepLabel.setForeground(java.awt.Color.black);
stepLabel.setFont(new Font("Dialog", Font.BOLD, 18));
stepLabel.setBounds(292,6,280,24);
}
}

```

F. CASES.GUI.UTIL PACKAGE

1. Cases.GUI.util.NextVersion

```

/**-----
 * Filename: NextVersion.java
 * @Date: 4-24-2003
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: this class retrieves the next version which is used by
 * objects to traverse from one version to the next (e.g. bridge step operations)
 *-----
 */
package Cases.GUI.util;
import Cases.GUI.avc.AVCOpenStepFrame;

```

```

/**
 * this class retrieves the next version
 */
public class NextVersion {
    /**
     * constructor which gathers information from avc.
     * information is dependent upon stepIDclicked and versionNumber
     * @param AVCOpenStepFrame, String, and String.
     */
    public NextVersion(AVCOpenStepFrame avc, String stepIDclicked, String versionNumber) {
        avc.getEHLComboBox().setSelectedIndex(1); // assumes only one EHL
        for (int s_count=1; s_count<avc.getStepIDComboBox().getItemCount(); s_count++){
            String tempString = (String)avc.getStepIDComboBox().getItemAt(s_count);
            if (stepIDclicked.equals(tempString)) {
                // put version info here
                for (int v_count=0; v_count<avc.getVersionComboBox().getItemCount(); v_count++) {
                    if (versionNumber.equals(avc.getVersionComboBox().getItemAt(v_count))) { // found the version
                        if (v_count < avc.getVersionComboBox().getItemCount()-1) { // is there a next version?
                            setNextVersion(avc.getVersionComboBox().getItemAt(v_count+1).toString());
                        } else { // if there isn't a next version return "END"
                            setNextVersion("END");
                        }
                    }
                }
            }
        }
    }

    /**
     * returns the next version
     * @return String
     */
    public String getNextVersion() { return nextVersion; }

    /**
     * sets the nextVersion
     * @param String
     */
    public void setNextVersion(String nextVersion) { this.nextVersion = nextVersion; }

    /** variable to track the next version */
    private String nextVersion;
}

```

2. Cases.GUI.util.PreviousVersion

```

/**-----
 * Filename: NextVersion.java
 * @Date: 4-24-2003
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: this class retrieves the previous version which is used by
 * objects to traverse from one version to the next (e.g. bridge step operations)
 *-----
 */
package Cases.GUI.util;
import Cases.GUI.avc.AVCOpenStepFrame;
/**
 * this class retrieves the previous version
 */
public class PreviousVersion {
    /**
     * constructor which gathers information from avc.
     * information is dependent upon stepIDclicked and versionNumber
     * @param AVCOpenStepFrame, String, and String.
     */
}

```

```

public PreviousVersion(AVCOpenStepFrame avc, String stepIDclicked, String versionNumber) {
    avc.getEHLComboBox().setSelectedIndex(1); // assumes only one EHL
    for (int s_count=1; s_count<avc.getStepIDComboBox().getItemCount(); s_count++){
        String tempString = (String)avc.getStepIDComboBox().getItemAt(s_count);
        if (stepIDclicked.equals(tempString)) {
            // put version info here
            for (int v_count=avc.getVersionComboBox().getItemCount();v_count>=0; v_count--) {
                if (versionNumber.equals(avc.getVersionComboBox().getItemAt(v_count))) { // found the version
                    if (v_count > 0) { // is there a previous version?
                        setPreviousVersion(avc.getVersionComboBox().getItemAt(v_count-1).toString());
                    } else {
                        setPreviousVersion("END");
                    }
                }
            }
        }
    }
}

/**
 * returns the previous version
 * @return String
 */
public String getPreviousVersion(){ return previousVersion; }

/**
 * sets the previousVersion
 * @param String
 */
public void setPreviousVersion(String previousVersion){ this.previousVersion = previousVersion; }

/** variable to track the previous version */
private String previousVersion;
}

```

3. Cases.GUI.util.Tools

```

/**-----
 * @Filename: Tools.java
 * @Date: 3-7-2003
 * @Author: Arthur Clomera for NPS Thesis
 * @Compiler: JDK 1.3.1
 * Description: some tool used by GUI subsystem
 * -----
 */

package Cases.GUI.util;

/**
 * a class with tools used by GUI subsystem
 */
public class Tools {
    /**
     * default constructor to access methods and attributes
     */
    public Tools() { }

    /**
     * a function to convert i to a string
     * @param int
     * @return String
     */
    public String intToString(int i) {
        // start names off with the letter 'a' + their array index
        char c=(char)((int)'a'+i);
        return ""+c;
    }
}

```



```

    }

    /**
     * StringToInt will take a string of format aComponent and return the int value of the first char.
     * therefore aComponent returns 0
     * @param String
     * @return int
     */
    public int StringToInt(String s) {
        char[] c = s.substring(0,1).toCharArray();
        return (int)c[0]-97;
    }

    /**
     * a main procedure for unit testing
     */
    public static void main (String[] args) {
        Tools tool = new Tools();
        for (int i=0;i<5;i++) {
            System.out.print (tool.intToString(i)+" ");
            System.out.println (tool.StringToInt(tool.intToString(i)));
        }
    }
}

```

G. CASES.INTERFACES PACKAGE

1. Cases.Interfaces.I_AVC

```

/**-----
 * Filename: I_AVC.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Moved by: Arthur B. Clomera to Cases.Interfaces Package
 * Compiler: JDK 1.3.1
 * Description: an interface for 3 classes - AVCCreateStepFrame
 *              - AVCMergingFrame
 *              - AVCSplittingFrame
 *-----
 */
package Cases.Interfaces;

import java.io.File;
import java.util.Vector;
import Cases.Dependency;

////////////////////////////////////
/**
 * I_AVC : an interface for 3 classes - AVCCreateStepFrame
 *              - AVCMergingFrame
 *              - AVCSplittingFrame
 */
////////////////////////////////////
public interface I_AVC {
    void checkPath(Vector v);
    void createFiles( Dependency dep, File theFile);
    void setLoopNameComboBox( Vector loopVector );
    Vector tokenizeVector(String string);
}

```

2. Cases.Interfaces.I_AVCOpenStep

```

/**-----
 * Filename: I_AVCOpenStep.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Moved by: Arthur B. Clomera to Cases.Interfaces Package
 * Compiler: JDK 1.3.1
 * Description: an interface for AVCOpenStepFrame
 *-----
 */
package Cases.Interfaces;

import java.util.Vector;
import Cases.VersionControl;

////////////////////////////////////
/**
 * I_AVCOpenStep : an interface for AVCOpenStepFrame
 */
////////////////////////////////////
public interface I_AVCOpenStep{
    void setInitialFrame( VersionControl vc );
    void setLoopNameComboBox( Vector loopVector );
    void setStepComboBox( Vector stepVector );
    void setVersionComboBox( Vector versionVector );
}

```

3. Cases.Interfaces.I_Cases

```

/**-----
 * Filename: I_Cases.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Moved by: Arthur B. Clomera to Cases.Interfaces Package
 * Compiler: JDK 1.3.1
 * Description: an interface for CasesFrame
 *-----
 */
package Cases.Interfaces;

import java.io.File;
import java.util.Vector;
import Cases.TraceFrame;

////////////////////////////////////
/**
 * I_Cases: an interface for CasesFrame
 */
////////////////////////////////////
public interface I_Cases{
    void getVersionControl();
    void directoryTree();
    void currentDir(String absPath);
    void subDir(String absPath);
    Vector getComponents(String stepName, String absPath);
    void setComponentContent(String stepName, String absPath);
    void setStepContent(TraceFrame traceFrame, String stepName, String absPath);
    void setTraceFrame(String stepName, String absPath);
    void getAtomics(File aFile, Vector storedVector);
    int findMax(String thePath);
    void tokenizer(Vector v, String s);
    Vector fileNameList();
    void savePersonnelVector(Vector v);
    Vector getPersonnelVector();
    Vector getAllAtomics();
    void saveStepContentVector(Vector v, Vector majorMinorJobs, String stepName);
}

```

```

    Vector getStepContentVector();
    Vector getScheduledAtomicVector();
}

```

4. Cases.Interfaces.I_ComponentContent

```

/**-----
 * Filename: I_ComponentContent
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Moved by: Arthur B. Clomera to Cases.Interfaces Package
 * Compiler: JDK 1.3.1
 * Description: an interface for ComponentContentFrame
 *-----
 */
package Cases.Interfaces;

import java.io.File;
import java.util.Vector;
import javax.swing.*;

////////////////////////////////////
/**
 * I_ComponentContent: an interface for ComponentContentFrame
 */
////////////////////////////////////
public interface I_ComponentContent{
    String checkInput(String selectedItem);
    JComboBox findComboBox(String fileType);
    void saveLinkFile(String fileType);
    void searchFiles(File aFile, String fileType);
    File searchPath(String s);
    void setTextLink(Vector v);
    void setWordLink(Vector v);
    void setExcelLink(Vector v);
    void setDataLink(Vector v);
    void setURLLink(Vector v);
    void setCAPSLink(Vector v);
}

```

5. Cases.Interfaces.I_EditDecompose

```

/**-----
 * Filename: I_EditDecompose
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Moved by: Arthur B. Clomera to Cases.Interfaces Package
 * Compiler: JDK 1.3.1
 * Description: an interface for EditDecomposeFrame
 *-----
 */
package Cases.Interfaces;

import java.io.File;

////////////////////////////////////
/**
 * I_EditDecompose: an interface for EditDecomposeFrame
 */
////////////////////////////////////
public interface I_EditDecompose{
    boolean checkInputs(String secondary);
    void copyFile(File oldFile, File newFile);
}

```

6. Cases.Interfaces.I_Personnel

```
/**-----
 * Filename: I_Personnel
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Moved by: Arthur B. Clomera to Cases.Interfaces Package
 * Compiler: JDK 1.3.1
 * Description: an interface for PersonnelFrame
 *-----
 */
package Cases.Interfaces;

import java.util.Vector;

import Cases.Personnel;

////////////////////////////////////
/**
 * I_Personnel : an interface for PersonnelFrame
 */
////////////////////////////////////
public interface I_Personnel {
    void setInitial(Personnel personnel);
    Personnel getPersonnelData();
    void setSkillComboBox( Vector v);
}
}
```

7. Cases.Interfaces.I_ProjectSchema

```
/**-----
 * Filename: I_ProjectSchema
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Moved by: Arthur B. Clomera to Cases.Interfaces Package
 * Compiler: JDK 1.3.1
 * Description: an interface for ProjectSchemaFrame
 *-----
 */
package Cases.Interfaces;

import java.util.Vector;
import Cases.ComponentType;
import Cases.EHL;

////////////////////////////////////
/**
 * I_ProjectSchema: an interface for ProjectSchemaFrame
 */
////////////////////////////////////
public interface I_ProjectSchema {
    void readDepFile();
    void readInputFiles(String project);
    void setCompComboBox(Vector compVector);
    void setCompInfo(ComponentType ct);
    void setDepenEHLComboBox(Vector EHLVector);
    void setDependStepComboBox(Vector stepVector);
    void setDepInfo(String selectedDepStep);
    void setEHLComboBox(Vector stepVector);
    void setEHLInfo(EHL ehl);
    void setListModel(Vector stepVector);
    void setStepComboBox(Vector setpVector);
}
```

```

    void setItemList(Object[] objs);
}

```

8. Cases.Interfaces.I_StepContent

```

/**-----
 * Filename: I_StepContent
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Moved by: Arthur B. Clomera to Cases.Interfaces Package
 * Compiler: JDK 1.3.1
 * Description: an interface for StepContentFrame
 *-----
 */
package Cases.Interfaces;

import java.util.Vector;
import Cases.StepContent;

////////////////////////////////////
/**
 * I_StepContent: an interface for StepContentFrame
 */
////////////////////////////////////
public interface I_StepContent{
    StepContent getStepContent();
    void setInitial(StepContent stepContent);
    void setPredecessorComboBox(Object[] oa);
    void setReadOnly();
    void setStakeHolders();
    void setSkillComboBox(Vector v);
}

```

9. Cases.Interfaces.I_Trace

```

/**-----
 * Filename: I_Trace
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Moved by: Arthur B. Clomera to Cases.Interfaces Package
 * Compiler: JDK 1.3.1
 * Description: an interface for TraceFrame
 *-----
 */
package Cases.Interfaces;

import java.io.File;
import java.util.Vector;

////////////////////////////////////
/**
 * I_Trace : an interface for TraceFrame
 */
////////////////////////////////////
public interface I_Trace{
    String checkSelection(String selectedItem );
    String convertToThePath(String s);
    File searchFilePath( String s);
    void searchFiles(File aFile, String fileType);
    void searchIndex();
    void searchInputFiles( String thePath );
    void searchPath( String s);
}

```

```

        void setComponentContent(String selectedItem, File f);
void setHistory( String s);
void setInitial(Vector componentsVector);
void setSelectedItem(String currentPath, String selectedItem);
        void tokenizer( Vector v, String s);
}

```

H. CASE.QFD PACKAGE

1. Cases.QFD.CalcMatrix

```

/**-----
 * Filename: CalcMatrix.java
 * @Date: 1-29-2003
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: CalcMatrix provides the functionality to do a upstream
 * or downstream calculation on a House of Quality (HOQ).
 * This object users the following terminology. left - "what" matrix, right - "how" matrix,
 * middle - correlation matrix in the HOQ.
 *-----
 */

package Cases.QFD;

import java.io.Serializable;

import Cases.QFD.util.GetDoubleValue;

/**
 * CalcMatrix provides the functionality to do a upstream or downstream calculation
 * on a House of Quality
 */
public class CalcMatrix implements Serializable {
    /**
     * Polymorphic method CalcMatrix for single left matrix
     */
    public CalcMatrix(MyDefaultTableModel left, MyDefaultTableModel top, MyDefaultTableModel middle) {
        super();
        this.leftMatrix = left;
        this.topMatrix = top;
        this.depMatrix = middle;
        this.vectorSize=1;
    }
    /**
     * Polymorphic method calcMatrix for multiple left matrices
     */
    public CalcMatrix(MyDefaultTableModel left, MyDefaultTableModel top, MyDefaultTableModel middle, int vectorSize) {
        super();
        this.leftMatrix = left;
        this.topMatrix = top;
        this.depMatrix = middle;
        this.vectorSize = vectorSize;
    }
    /**
     * CalcButtonActionPerformed: calculates the top matrix
     * calculations go from left matrix through middle matrix and solution
     * placed in top matrix.
     * @return void
     * @param void
     */
    public void forwardCalculation() {
        /** an object to do double value operations */
        GetDoubleValue gdv = new GetDoubleValue();
        /** the number of rows */

```

```

int row = leftMatrix.getRowCount();
/** the number of columns */
int col = depMatrix.getColumnCount();
/** a variable to store column sums */
double[] sum= new double[col];
/** a variable to store the sum of left column (weight) */
double weight = 0.0;
/** a variable to total all sums */
double total = 0.0;
// get the weight according to the left matrix
for (int i = 0; i<row; i++) {
    weight += gdv.getValue(leftMatrix.getValueAt(i,2));
}

for (int i = 0; i<col; i++) {
    sum[i]=0.0; // initialize column sums
    for (int j = 0; j<row; j++) {
        double amount=0.0; // initialize amount to 0.0
        try { // the matrix value should be a double value
            amount = gdv.getValue(depMatrix.getValueAt(j,i));
        }
        catch (java.lang.NumberFormatException e2) {
            System.out.println("Value error in dependency Matrix");
        }
        // sum up each row by multiplying the amount in the matrix
        // by the value in the left matrix
        sum[i] += amount * gdv.getValue(leftMatrix.getValueAt(j,2));
    }
    // total up the sums
    total += sum[i];
}

// normalize sums
for (int i =0; i<col; i++) {
    sum[i] /= total;
    // get ratio to other sums
    sum[i] *= weight;
    // normalize to the weight of the left elements
    double temp = (gdv.roundUp(sum[i]))/this.vectorSize; // round result up and format for output
    topMatrix.setValueAt(new Double(temp),2,i);
}
}
/**
 * revCalcButtonActionPerformed: calculates the left matrix
 * calculations go from top matrix through middle matrix and solution
 * placed in left matrix.
 * @return void
 * @param void
 */
public void reverseCalculation() {
    /** an object to do double value operations */
    GetDoubleValue gdv = new GetDoubleValue();
    /** the number of rows */
    int row = leftMatrix.getRowCount();
    /** the number of columns */
    int col = depMatrix.getColumnCount();
    /** array for sum for each row */
    double[] sum= new double[row];
    /** a variable to store the sum of left column (weight) */
    double weight = 0.0;
    /** a variable to total all sums */
    double total = 0.0;
    // get the weight according to the top matrix
    for (int i = 0; i<col; i++) {
        weight += gdv.getValue(topMatrix.getValueAt(2,i));
    }
}

```

```

for (int i = 0; i < row; i++) {
    sum[i] = 0.0; // initialize the row sums
    for (int j = 0; j < col; j++) {
        double amount = 0.0; // initialize amount to 0.0
        try { // values in matrix should be double values
            amount = gdv.getValue(depMatrix.getValueAt(i, j));
        }
        catch (java.lang.NumberFormatException e2) {
            System.out.println("Value error in dependency Matrix");
        }

        // sum up each column by multiplying the amount in the matrix
        // by the value in the top matrix
        sum[i] += amount * gdv.getValue(topMatrix.getValueAt(2, j));
    }

    total += sum[i];
}

// normalize sums
for (int i = 0; i < row; i++) {
    sum[i] /= total;
    // get ratio to other sums
    sum[i] *= weight;
    // normalize to the weight of the left elements
    double temp = (gdv.roundUp(sum[i])) / this.vectorSize; // round result up and format for output
    leftMatrix.setValueAt(new Double(temp), i, 2);
}
}

/** the left-most ("what") matrix of the house of quality */
private MyDefaultTableModel leftMatrix;
/** the top-most ("how") matrix of the house of quality */
private MyDefaultTableModel topMatrix;
/** the middle ("correlation") matrix of the house of quality */
private MyDefaultTableModel depMatrix;
/** variable to store vector size */
private int vectorSize = 1;
}

```

2. Cases.QFD.CombinedHouseMatrix

```

/**-----
 * Filename: HouseMatrix.java
 * @Date: 10-29-2002
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: This class displays the QFD house matrix or roof matrix.
 * This object uses the following terminology. left - "what" matrix, right - "how" matrix,
 * middle - correlation matrix in the HOQ.
 *-----
 */
package Cases.QFD;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.Vector;
import javax.swing.*;
import Cases.GUI.util.Tools;

```



```

import Cases.QFD.util.MatrixCalculator;

/**
 * @testcase test.Cases.QFD.TestHouseMatrix
 */
public class CombinedHouseMatrix extends JFrame implements Serializable, ActionListener {
    // table models to hold QFD information
    private MyDefaultTableModel leftElements;
    private MyDefaultTableModel dependencyElements;
    private MyDefaultTableModel topElements;
    private MyDefaultTableModel oldTopElements;
    // scroll pane for each matrix
    private JScrollPane leftScrollPane;
    private JScrollPane depScrollPane;
    private JScrollPane topScrollPane;
    // represent each matrix with a JTable
    private JTable topTable;
    private JTable depTable;
    private JTable leftTable;
    /** a string for the project name */
    private String projectName;
    // house has a menu of file operations and views
    private JMenu fileMenu = new JMenu("File");
    private JMenuBar bar = new JMenuBar();
    private JMenuItem openQFDMenuItem = new JMenuItem("Open QFD");
    private JMenuItem saveQFDMenuItem = new JMenuItem("Save QFD");
    private JMenuItem deleteQFDMenuItem = new JMenuItem("Delete QFD");
    private JMenu viewMenu = new JMenu("View");
    private JMenuItem userDefinedViewMenuItem = new JMenuItem("Dependency Threshold");
    private JMenuItem traceViewMenuItem = new JMenuItem("Component Trace");
    /** a variable to hold dependency index */
    private int depIndex;
    /** component id of origin */
    private String origin;
    /** vector of left ids */
    private Vector leftID;
    /** component id of top matrix */
    private String topID;
    /** size of vector */
    private int vectorSize;
    /** boolean to track if this representation is a house (true) or roof (false). */
    private boolean isHouse = true;
    /** flag to tell object to save house and a change has been made */
    public boolean saveHouse = false;

    /**
     * Initiates the 'Quit' event
     */
    private JMenuItem exitMenuItem = new JMenuItem("Exit");
    private JButton exitButton;
    private JButton calcButton;
    private int numOfComponents;
    private String houseVersion;

    /**
     * HouseMatrix: constructor (polymorphic)
     * @return void
     * @param void
     */
    public CombinedHouseMatrix() {
        super("Quality Function Deployment (QFD) House of Quality");
    }

    /**
     * CombinedHouseMatrix: constructor (polymorphic)
     * CombinedHouseMatrix can have multiple secondary input components which are stored in left vector
     * along with there dependency information which is stored in the middle vector.
     * displays combined house (with out default value)
     * @return void

```

```

* @param String, 2 Vectors of DefaultTableModels and 1 DefaultTableModels
**/
public CombinedHouseMatrix(String projectName, String dep, int depNumber, int compNumber, String originLocation,
    Vector left, MyDefaultTableModel top, Vector middle,
    Vector c1Name, String c2Name, String currentVersion) {
    super ("Quality Function Deployment (QFD) House of Quality for "+dep+" (Origin)");
    this.projectName = projectName;
    this.houseVersion = currentVersion;
    this.leftID = c1Name;
    this.topID = c2Name;
    this.isHouse = true;
    this.vectorSize = left.size();
    this.oldTopElements = top;
    depIndex = depNumber;
    this.numOfComponents = compNumber;
    this.origin = originLocation;
    int row = ((MyDefaultTableModel)left.get(0)).getRowCount(); // initialize with first
element
    leftElements = ((MyDefaultTableModel)left.get(0)).makeClone(); // initialize with first element
    leftElements.makeClone();
    dependencyElements = ((MyDefaultTableModel)middle.get(0)).makeClone(); // initialize with first element
    for (int i=1;i<vectorSize; i++) {
        // initialize with other elements
        row += ((MyDefaultTableModel)left.get(i)).getRowCount();
        leftElements.join(((MyDefaultTableModel)left.get(i)).makeClone());
        dependencyElements.join(((MyDefaultTableModel)middle.get(i)).makeClone());
    }
    MatrixCalculator mc = new MatrixCalculator();
    topElements = mc.transposeComponent(top);
    int col = topElements.getColumnCount();
    // can not modify dependencies in a combined house matrix
    //dependencyElements.setOrigin(true);
    //middle.set(0,dependencyElements);
    initGUI();
    // check to see if this is a user-defined view
    if (depIndex == -1) {
        calcButton.hide();
        exitButton.hide();
    }
}
/**
* CombinedHouseMatrix: constructor (polymorphic)
* CombinedHouseMatrix can have multiple secondary input components which are stored in left vector
* along with there dependency information which is stored in the middle vector.
* displays combined house (with default value)
* @return void
* @param String, 2 vector of DefaultTableModels, 1 DefaultTableModels, Object
**/
public CombinedHouseMatrix(String projectName, String dep, int depNumber, int compNumber, String originLocation, Vector
left, MyDefaultTableModel top, Vector middle, Object defaultValue, Vector c1Name, String c2Name, String currentVersion) {
    super ("Quality Function Deployment (QFD) House of Quality for "+dep+" (Default)");
    this.projectName = projectName;
    this.houseVersion = currentVersion;
    isHouse = true;
    depIndex = depNumber;
    this.leftID = c1Name;
    this.topID = c2Name;
    this.oldTopElements = top;
    this.numOfComponents = compNumber;
    this.origin = originLocation;
    this.vectorSize = left.size();
    int row = ((MyDefaultTableModel)left.get(0)).getRowCount(); // initialize with first element
    leftElements = ((MyDefaultTableModel)left.get(0)).makeClone(); // initialize with first element
    dependencyElements = ((MyDefaultTableModel)middle.get(0)).makeClone(); // initialize with first element
    for (int i=1;i<vectorSize; i++) { // initialize with other
elements
        row += ((MyDefaultTableModel)left.get(i)).getRowCount();
        leftElements.join(((MyDefaultTableModel)left.get(i)).makeClone());

```

```

        dependencyElements.join(((MyDefaultTableModel)middle.get(i)).makeClone());
    }
    for (int i=0; i<row; i++)
        leftElements.setValueAt(defaultValue,i,2);
    MatrixCalculator mc = new MatrixCalculator();
    topElements = mc.transposeComponent(top);

    int col = topElements.getColumnCount();
    for (int j=0; j<col; j++)
        topElements.setValueAt(defaultValue,2,j);
        // can not modify dependencies in a combined house matrix
        //dependencyElements.setOrigin(true);

    initGUI();
}

/**
 * initGUI: procedure to initialize the QFD GUI
 * @return void
 * @param void
 */
public void initGUI() {
    // calc button
    calcButton = new JButton();
    calcButton.setText("jButton1");
    calcButton.setActionCommand("calcButton");
    calcButton.setLabel("Calc");
    calcButton.setToolTipText("calculate dependencies");
    calcButton.setIcon(new javax.swing.ImageIcon("C:/CASES/IMAGES/REDO.GIF"));
    calcButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    calcButton.setFont(new java.awt.Font("SansSerif", java.awt.Font.BOLD, 14));
    calcButton.addActionListener(this);

    // exit button
    exitButton = new JButton();
    exitButton.setText("exitButton");
    exitButton.setActionCommand("exitButton");
    exitButton.setLabel("Save & Exit");
    exitButton.setToolTipText("save QFD and exit HOQ");
    exitButton.setMnemonic('X');
    exitButton.setFont(new java.awt.Font("SansSerif", java.awt.Font.BOLD, 12));
    exitButton.addActionListener(this);

    // file menu : with Open, Save, Delete, and Exit menu items
    fileMenu.setActionCommand("File");
    fileMenu.setMnemonic((int)'F');
    openQFDMenuItem.setActionCommand("Open QFD");
    openQFDMenuItem.setMnemonic((int)'O');
    openQFDMenuItem.addActionListener(this);

    fileMenu.add(openQFDMenuItem);
    saveQFDMenuItem.setActionCommand("Save QFD");
    saveQFDMenuItem.setMnemonic((int)'S');
    saveQFDMenuItem.addActionListener(this);
    fileMenu.add(saveQFDMenuItem);
    deleteQFDMenuItem.setActionCommand("Delete QFD");
    deleteQFDMenuItem.setMnemonic((int)'D');
    deleteQFDMenuItem.addActionListener(this);
    fileMenu.add(deleteQFDMenuItem);
    fileMenu.addSeparator ();
    exitMenuItem.setActionCommand("Exit");
    exitMenuItem.setMnemonic((int)'X');
    exitMenuItem.addActionListener(this);
    fileMenu.add(exitMenuItem);

    // view menu : with user-defined and trace menu items
    viewMenu.setActionCommand("View");
    viewMenu.setMnemonic((int)'V');
    userDefinedViewMenuItem.setActionCommand("User-Defined");
    userDefinedViewMenuItem.setMnemonic((int)'D');
    userDefinedViewMenuItem.addActionListener(this);

```

```

        viewMenu.add (userDefinedViewMenuItem);
traceViewItem.setActionCommand("Trace");
traceViewItem.setMnemonic((int) 'C');
traceViewItem.addActionListener(this);
viewMenu.add(traceViewItem);

        setJMenuBar(bar);
        bar.add(fileMenu);
        bar.add(viewMenu);
// setup left matrix
        leftTable = new JTable(leftElements);
        leftTable.setPreferredScrollableViewportSize(new Dimension(500, 70));
// setup middle matrix
        depTable = new JTable(dependencyElements);
        depTable.setPreferredScrollableViewportSize(new Dimension(500, 70));
        depTable.setBackground(Color.yellow);
// setup top matrix
        topTable = new JTable(topElements);
        topTable.setPreferredScrollableViewportSize(new Dimension(500, 70));
// Create a scroll pane fore each JTable
        topScrollPane = new JScrollPane();
        depScrollPane = new JScrollPane();
        leftScrollPane = new JScrollPane();
        // Create a new table instance
        getContentPane().setLayout(new java.awt.GridBagLayout());
        getContentPane().add(calcButton, new
java.awt.GridBagConstraints(0,2,1,1,0.0,0.0,java.awt.GridBagConstraints.CENTER,java.awt.GridBagConstraints.NONE,new
java.awt.Insets(0,0,0,0),0,0));
        getContentPane().add(exitButton, new
java.awt.GridBagConstraints(0,1,1,1,0.0,0.0,java.awt.GridBagConstraints.CENTER,java.awt.GridBagConstraints.NONE,new
java.awt.Insets(0,0,0,0),0,0));
        getContentPane().add(depScrollPane,
        new java.awt.GridBagConstraints(1, 3, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.NONE,
        new java.awt.Insets(0, 0, 0, 0), 0, 0));
        getContentPane().add(leftScrollPane,
        new java.awt.GridBagConstraints(0, 3, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.NONE,
        new java.awt.Insets(0, 0, 0, 0), 0, 0));
        getContentPane().add(topScrollPane,
        new java.awt.GridBagConstraints(1, 2, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(0, 0, 3, 0), 0, 0));
// let left matrix scroll
        leftScrollPane.setPreferredSize(new java.awt.Dimension(300,300));
        leftScrollPane.setSize(new java.awt.Dimension(300,300));
        leftScrollPane.setMinimumSize(new java.awt.Dimension(300, 300));

        leftScrollPane.setHorizontalScrollBarPolicy(javax.swing.JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        leftScrollPane.setVerticalScrollBarPolicy(javax.swing.JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
// let middle matrix scroll
        depScrollPane.setMinimumSize(new java.awt.Dimension(500, 300));
        depScrollPane.setPreferredSize(new java.awt.Dimension(500, 300));
        depScrollPane.setViewportBorder(null);
        depScrollPane.setVerticalScrollBarPolicy(javax.swing.JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

        depScrollPane.setHorizontalScrollBarPolicy(javax.swing.JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
// let top matrix scroll
        topScrollPane.setMinimumSize(new java.awt.Dimension(300, 100));
        topScrollPane.setPreferredSize(new java.awt.Dimension(300, 100));
        topScrollPane.setSize(new java.awt.Dimension(300, 100));
        topScrollPane.setVerticalScrollBarPolicy(javax.swing.JScrollPane.VERTICAL_SCROLLBAR_NEVER);

        topScrollPane.setHorizontalScrollBarPolicy(javax.swing.JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        topScrollPane.getViewPort().add(topTable);
// top matrix customizations
        topTable.setBounds(new java.awt.Rectangle(21,25,50,50));
        topTable.setBorder(javax.swing.BorderFactory.createEtchedBorder());

```

```

        topTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
// middle matrix customizations
depScrollPane.getViewport().add(depTable);
depTable.setBounds(new java.awt.Rectangle(14,20,50,50));
depTable.setBorder(javax.swing.BorderFactory.createEtchedBorder());
depTable.setCellSelectionEnabled(true);
depTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
// left matrix customizations
leftScrollPane.getViewport().add(leftTable);
leftTable.setBounds(new java.awt.Rectangle(22,18,50,50));
leftTable.setBorder(javax.swing.BorderFactory.createEtchedBorder());
leftTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_ALL_COLUMNS);
setResizable(true);
setSize(new java.awt.Dimension(650, 700));

this.pack();
        this.setVisible(true);
        setBounds(new java.awt.Rectangle(0, 0, 822, 516));
    }
}
/**
 * saveQFD: save this class to a file
 * @return void
 * @param void
 */
public void saveQFD () {
    // create a SAVE FileDialog
    FileDialog fileDialog = new FileDialog(this,"Save", FileDialog.SAVE);
    fileDialog.setDirectory(projectName);
    fileDialog.show();

    // read the users file choice
    String dir = fileDialog.getDirectory();
    String file = fileDialog.getFile();
    // save file with .cfg extension
    if (file.indexOf("qfd") == -1) {
        if (file.indexOf(".") == -1)
            file = file + ".qfd";
        else
        {
            if (file.substring(file.length() -1, file.length()) == ".")
                file = file + "qfd";
        }
    }

    try {
        FileOutputStream f = new FileOutputStream(dir + file);
        ObjectOutputStream s = new ObjectOutputStream(f);
        s.writeObject(this);
        s.flush();
    }
    catch (Exception e) {
        System.out.println(e);
    }
}
/**
 * openQFD: open a file of an existing QFD matrix of class HouseMatrix
 * @return void
 * @param void
 */
public void openQFD () {
    // create a LOAD FileDialog
    FileDialog fileDialog = new FileDialog (this,"Open", FileDialog.LOAD);
    fileDialog.show();

    // what file and from which directory did the user select?
    String dir = fileDialog.getDirectory();
    String file = fileDialog.getFile();
    try {
        FileInputStream f = new FileInputStream (dir + file);
        ObjectInputStream s = new ObjectInputStream(f);

```

```

HouseMatrix house = (HouseMatrix)s.readObject();
// switch to new house

house.show();
house.repaint();
this.dispose();

}
catch (Exception e) {
    System.out.println(e);
}
}

    public void calcButtonActionPerformed () {
CalcMatrix cm = new CalcMatrix(leftElements, topElements, dependencyElements, this.vectorSize);
Tools tempTools = new Tools();
if (tempTools.StringToInt((String)leftID.get(0)) >= tempTools.StringToInt(this.origin))
    cm.forwardCalculation();
else
    cm.reverseCalculation();
    saveButtonActionPerformed();
}

public void saveButtonActionPerformed () {
    if (isHouse) { for (int i=0; i<this.oldTopElements.getRowCount(); i++)
        this.oldTopElements.setValueAt(topElements.getValueAt(2,i),i,2);
        this.saveHouse=true;
    }
}

}
/**
 * Action event handler for the menu events.
 *
 * @param e The action event that is created by selecting
 * a menu item from this menu
 */
    public void actionPerformed (ActionEvent event) {
        MatrixCalculator mc = new MatrixCalculator();
        Dialog dialog = new Dialog(this,true); // ensure the view closes before proceeding
        Object object = event.getSource();
        if (object == openQFDMenuItem) // open a saved QFD HOQ
            openQFD();
        else if (object == saveQFDMenuItem) // save a QFD HOQ
            saveQFD();
        else if (object == userDefinedViewMenuItem) { // user wants to the user-defined view
            double nTimes; // need a variable for 'n'
            boolean flagValue ; // need a flag for greater than or less than
            UserDefinedViewGUI newView = new UserDefinedViewGUI(dialog);
            nTimes = Double.parseDouble(newView.getNValue());
            flagValue = newView.getIsGreaterThan(); // > = true <= false
            new UserDefinedView (projectName, houseVersion, leftElements, mc.transposeComponent(topElements),
dependencyElements, nTimes, flagValue);
            newView.dispose();
        }
        else if (object == traceViewMenuItem) { // user wants to conduct a trace view
            int colSelected; // need to know the col or row to trace on
            double viewValue; // need to know the value to filter on
            TraceViewGUI traceView = new TraceViewGUI(dialog, topElements);
            viewValue = Double.parseDouble(traceView.getFilterValue());
            colSelected = traceView.getColumnSelected();
            // trace upstream
            new TraceUpstream (projectName, houseVersion, topID, depIndex, viewValue, colSelected);
            // trace downstream
            new TraceDownstream (projectName, houseVersion, topID, numOfComponents, depIndex, viewValue, colSelected);
        }
        else if (object == deleteQFDMenuItem) // user wants to delete a saved QFD HOQ
            System.out.println("Use DOS to delete QFD files...");
        else if (object == exitMenuItem) // user wants to close window
            this.dispose();
        else if (object == calcButton) { // user pressed calc button

```

```

        calcButtonActionPerformed();
    }
    else if (object == exitButton) {    // user pressed save and exit button
        saveButtonActionPerformed ();
        this.dispose();
    }
}

/**
 * main: main procedure for class testing
 * @return void
 * @param String[] args -- but not used
 */
public static void main (String[] args) {
    Object[] names = {"", "", ""};
    Object[] depNames = {"", "", "", ""};
    Object value = new Integer(2);
    // QFD upstream example or test data
    Object[][] tableData = {{ "R1", "good", "5"}, {"R2", "fast", "1"}, {"R3", "cheap", "3"} };
    Object[][] tableData2 = {{ "S1", "100%", "0.53"}, {"S2", "100mph", "1.06"}, {"S3", "$100.00", "4.76"}, {"S4", "100psi", "2.65"} };
    Object[][] tableDep = {{ "0", "1", "0", "3"}, {"3", "1", "0", "0"}, {"0", "0", "9", "0"} };

    String project = new String("MyProject");
    Vector t1 = new Vector();
    Vector matrix = new Vector();
    Vector namet1 = new Vector();
    t1.add( new MyDefaultTableModel (tableData,names));
    t1.add( new MyDefaultTableModel (tableData,names));
    namet1.add(new String("aComponent"));
    MyDefaultTableModel t2 = new MyDefaultTableModel (tableData2,names);
    matrix.add(new MyDefaultTableModel (tableDep, depNames));
    matrix.add(new MyDefaultTableModel (tableDep, depNames));
    ((MyDefaultTableModel)t1.get(0)).setOrigin(true);
    CombinedHouseMatrix house = new CombinedHouseMatrix(project,"Risk", 1, -1, (String)namet1.get(0), t1, t2, matrix, namet1,
    null, "0");
    house.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            System.exit(0);
        }
    } );
}
}

```

3. Cases.QFD.ComponentQFD

```

/**-----
 * Filename: ComponentQFD.java
 * @Date: 4-26-2003
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: the aggregation of QFD information for Component class
 *-----
 */

package Cases.QFD;

import java.io.Serializable;
import java.util.Vector;

public class ComponentQFD implements Serializable {
    /** compDepTable : stores the relationships between intra-components */
    private MyDefaultTableModel compDepTable;
    /** componentTable : stores component dependencies in a vector */
    private Vector componentTable = new Vector();
    /** a method to get a component table */

```

```

    public MyDefaultTableModel getComponentTable(int index){ return (MyDefaultTableModel) componentTable.get(index); }
    /** a method to modify component tables */
    public void setComponentTable(MyDefaultTableModel cTable, int index){ componentTable.setElementAt(cTable,index);}
    /** a method to add a new component table */
    public void addComponentTable(MyDefaultTableModel cTable) { componentTable.addElement(cTable);}
    /** a method to get the components dependency table */
    public MyDefaultTableModel getCompDepTable(){ return this.compDepTable; }
    /** a method to set the components dependency table */
    public void setCompDepTable(MyDefaultTableModel compDepTable){ this.compDepTable = compDepTable; }
}

```

4. Cases.QFD.HouseMatrix

```

/**-----
 * Filename: HouseMatrix.java
 * @Date: 10-29-2002
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: This class displays the QFD house matrix or roof matrix.
 * This object uses the following terminology. left - "what" matrix, right - "how" matrix,
 * middle - correlation matrix in the HOQ.
 * Note: the polymorphic constructor have very similar code but require different signatures.
 *-----
 */
package Cases.QFD;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import javax.swing.*;
import Cases.GUI.util.Tools;
import Cases.QFD.util.MatrixCalculator;

/**
 * This class displays the QFD house matrix or roof matrix.
 */
public class HouseMatrix extends JFrame implements Serializable, ActionListener {
    // table models needed for each matrix
    private MyDefaultTableModel leftElements;
    private MyDefaultTableModel dependencyElements;
    private MyDefaultTableModel topElements;
    // tables to keep track of old top and middle information
    private MyDefaultTableModel oldTopElements;
    private MyDefaultTableModel oldDepElements;
    // scroll each matrix
    private JScrollPane leftScrollPane;
    private JScrollPane depScrollPane;
    private JScrollPane topScrollPane;
    // each matrix is represented by a JTable
    private JTable topTable;
    private JTable depTable;
    private JTable leftTable;
    /** string to store project name */
    private String projectName;
    // a file menu with open, save, delete, and exit menu items
    private JMenu fileMenu = new JMenu("File");
    private JMenuBar bar = new JMenuBar();
    private JMenuItem openQFDMenuItem = new JMenuItem("Open QFD");

```



```

        private JMenuItem saveQFDMenItem = new JMenuItem("Save QFD");
        private JMenuItem deleteQFDMenItem = new JMenuItem("Delete QFD");
        private JMenuItem exitMenItem = new JMenuItem ("Exit");
        // a View menu with the dependency threshold and component trace views.
        private JMenu viewMenu = new JMenu("View");
        private JMenuItem userDefinedViewMenItem = new JMenuItem("Dependency Threshold");
        private JMenuItem traceViewMenItem = new JMenuItem("Component Trace");

        /** the dependency index */
        private int depIndex;
        /** the left id, top id, and origin (all component ids)*/
        private String leftID, topID, origin;
        /** a flag to track if HOQ represents a house (true) or roof (false) */
        private boolean isHouse = true;
        /** a flag to let object know if changes have been made that need to be saved */
        public boolean saveHouse = false;

        // buttons
        private JButton exitButton;
        private JButton calcButton;
        /** the number of components in the draw pane */
        private int numOfComponents;
        /** the version that the HOQ represents */
        private String houseVersion;

        /**
         * HouseMatrix: constructor (polymorphic)
         * @return void
         * @param void
         */
        public HouseMatrix() {
            super ("Quality Function Deployment (QFD) House of Quality");
        }

        /**
         * HouseMatrix: constructor (polymorphic)
         * HouseMatrix can display a single secondary input components (left table model)
         * along with its dependency (middle table model) information.
         * displays house (with out default value)
         * @return void
         * @param String and 3 DefaultTableModels
         */
        public HouseMatrix(String projectName, String dep, int depNumber, int compNumber, String originLocation,
            MyDefaultTableModel left, MyDefaultTableModel top, MyDefaultTableModel middle,
            String c1Name, String c2Name, String currentVersion) {
            super ("Quality Function Deployment (QFD) House of Quality for "+dep);
            this.projectName = projectName; // store project name
            this.houseVersion = currentVersion; // store current version
            this.leftID = c1Name; // store left component id
            this.topID = c2Name; // store top component id
            this.isHouse = true; // this is a house
            depIndex = depNumber; // store dependency number
            this.numOfComponents = compNumber; // store the number of components
            this.origin = originLocation; // store the origin component id
            int row = left.getRowCount(); // get the number of rows

            // assign matrice information
            leftElements = left;
            oldTopElements = top;
            oldDepElements = middle;
            // create an object to handle matrix calcuations
            MatrixCalculator mc = new MatrixCalculator();
            // transpose the top matrix
            topElements = mc.transposeComponent(top);

            int col = topElements.getColumnCount(); // get the number of columns

            // check to see row and col match up to middle matrix
            if (row==middle.getRowCount() && col==middle.getColumnCount())

```

```

        dependencyElements = middle;
    else { // they didn't so zero the matrix
        dependencyElements = new MyDefaultTableModel(row,col);
        dependencyElements.setColumnIdentifiers(topElements.getColumnIdentifiers());
        for (int i=0; i<row; i++)
            for (int j=0; j<col; j++)
                dependencyElements.setValueAt("0",i,j);
    }

    dependencyElements.setOrigin(true);
    middle = dependencyElements;
    initGUI();
    // check to see if this is a user-defined view
    if (depIndex == -1) {
        calcButton.hide();
        exitButton.hide();
    }
}
}
/**
 * HouseMatrix: constructor (polymorphic)
 * HouseMatrix can display a single secondary input components (left table model)
 * along with its dependency (middle table model) information.
 * displays house (with default value. see Object default value)
 * @return void
 * @param String, 3 DefaultTableModels, Object
 */
public HouseMatrix(String projectName, String dep, int depNumber, int compNumber, String originLocation,
                    MyDefaultTableModel left, MyDefaultTableModel top, MyDefaultTableModel middle,
                    Object default value, String c1Name, String c2Name, String currentVersion) {
    super ("Quality Function Deployment (QFD) House of Quality for "+dep);
    this.projectName = projectName; // store project name
    this.houseVersion = currentVersion; // store current version
    this.leftID = c1Name; // store left component id
    this.topID = c2Name; // store top component id
    this.isHouse = true; // this is a house
    depIndex = depNumber; // store dependency number
    this.numOfComponents = compNumber; // store the number of components
    this.origin = originLocation; // store the origin component id
    int row = left.getRowCount(); // get the number of rows

    // assign left matrix information
    leftElements = left;
    for (int i=0; i<row; i++) // assign the left matrix the default value
        leftElements.setValueAt(defaultValue,i,2);
    // create an object to perform matrix calculations
    MatrixCalculator mc = new MatrixCalculator();
    // assign matrix information
    oldTopElements = top;
    oldDepElements = middle;
    // transpose top matrix
    topElements = mc.transposeComponent(top);

    int col = topElements.getColumnCount(); // store the number of columns
    for (int j=0; j<col; j++) // assign the top matrix the default value
        topElements.setValueAt(defaultValue,2,j);
    // make sure that the row and col are the same size as the middle matrix
    if (row==middle.getRowCount() && col==middle.getColumnCount())
        dependencyElements = middle;
    else { // not the same therefore zero middle matrix
        dependencyElements = new MyDefaultTableModel(row,col);
        dependencyElements.setColumnIdentifiers(topElements.getColumnIdentifiers());
        for (int i=0; i<row; i++)
            for (int j=0; j<col; j++)
                dependencyElements.setValueAt("0",i,j);
    }

    dependencyElements.setOrigin(true);
    middle = dependencyElements;
    initGUI();
}
}
/**

```

```

        * HouseMatrix: constructor (polymorphic)
    * HouseMatrix can display a single components (left table model)
    * along with its dependency (middle table model) information.
    * display roof (without initial value)
    * @return void
    * @param String and 2 DefaultTableModels
    */
    public HouseMatrix(String projectName, String dep, MyDefaultTableModel left, MyDefaultTableModel middle, String
c1Name, String currentVersion) {
        super ("Quality Function Deployment (QFD) House of Quality for "+dep);
        isHouse = false; // this is a roof
        this.projectName = projectName; // store project name
        this.houseVersion = currentVersion; // store current version
        depIndex = -1; // user can not
calculate a roof
        this.leftID = c1Name; // store left component id
        int row = left.getRowCount(); // get the number of rows
        // assign left and middle matrice information
        leftElements = left;
        leftElements.setOrigin(false);
        oldDepElements = middle;
        // create an object to perform matrix calculations
        MatrixCalculator mc = new MatrixCalculator();
        // transpose left matrix to represent in top matrix
        topElements = mc.transposeComponent(left);
        int col = topElements.getColumnCount(); // store the number of columns
        // make sure that the row and col are the same size as the middle matrix
        if (row==middle.getRowCount() && col==middle.getColumnCount())
            dependencyElements = middle;
        else { // not the same therefore zero middle matrix
            dependencyElements = new MyDefaultTableModel(row,col);
            dependencyElements.setColumnIdentifiers(topElements.getColumnIdentifiers());
            for (int i=0; i<row; i++)
                for (int j=0; j<col; j++)
                    dependencyElements.setValueAt("0",i,j);
        }
        // allow the middle matrix to be modified to add drivers and indicators
        dependencyElements.setOrigin(true);
        middle = dependencyElements;
        initGUI();
        this.calcButton.hide();
    }

    /**
    * HouseMatrix: constructor (polymorphic)
    * HouseMatrix can display a single components (left table model)
    * along with its dependency (middle table model) information.
    * display roof with initial value (see Object defaultValue)
    * @return void
    * @param String, 2 DefaultTableModels, Object
    */
    public HouseMatrix(String projectName, String dep, MyDefaultTableModel left, MyDefaultTableModel middle,
        Object defaultValue, String c1Name, String currentVersion) {
        super ("Quality Function Deployment (QFD) House of Quality for "+dep);
        isHouse = false; // this is a roof
        this.projectName = projectName; // store project name
        this.houseVersion = currentVersion; // store current version
        depIndex = -1; // user can not
calculate a roof
        this.leftID = c1Name; // store left component id
        int row = left.getRowCount(); // get the number of rows
        // assign left and middle matrice information
        leftElements = left;
        leftElements.setOrigin(false);
        oldDepElements = middle;
        // initialize roof with default value
        for (int i=0; i<row; i++)
            leftElements.setValueAt(defaultValue,i,2);
        // an object to perform matrix calculations

```

```

MatrixCalculator mc = new MatrixCalculator();
// transpose left matrix and represent it in the top matrix
topElements = mc.transposeComponent(left);

int col = topElements.getColumnCount();          // store the number of columns
// populate the top matrix with the default value
for (int j=0; j<col; j++)
    topElements.setValueAt(defaultValue,2,j);
// make sure that the row and col are the same size as the middle matrix
if (row==middle.getRowCount() && col==middle.getColumnCount())
    dependencyElements = middle;
else { // size didn't match therefore zero the middle matrix
    dependencyElements = new MyDefaultTableModel(row,col);
    dependencyElements.setColumnIdentifiers(topElements.getColumnIdentifiers());
    for (int i=0; i<row; i++)
        for (int j=0; j<col; j++)
            dependencyElements.setValueAt("0",i,j);
}
// allow the middle matrix to be modified to add drivers and indicators
dependencyElements.setOrigin(true);
middle = dependencyElements;
initGUI();
this.calcButton.hide();
}
}
/**
 * initGUI: procedure to initialize the QFD GUI
 * @return void
 * @param void
 */
public void initGUI() {
    // calc button configuration
    calcButton = new JButton();
    calcButton.setText("jButton1");
    calcButton.setActionCommand("calcButton");
    calcButton.setLabel("Calc");
    calcButton.setToolTipText("calculate dependencies");
    calcButton.setIcon(new javax.swing.ImageIcon("C:/CASES/IMAGES/REDO.GIF"));
    calcButton.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    calcButton.setFont(new java.awt.Font("SansSerif", java.awt.Font.BOLD, 14));
    calcButton.addActionListener(this);

    // exit button configuration
    exitButton = new JButton();
    exitButton.setText("exitButton");
    exitButton.setActionCommand("exitButton");
    exitButton.setLabel("Save & Exit");
    exitButton.setToolTipText("save QFD and exit HOQ");
    exitButton.setMnemonic('X');
    exitButton.setFont(new java.awt.Font("SansSerif", java.awt.Font.BOLD, 12));
    exitButton.addActionListener(this);

    // file menu item with open, save, delete, and exit menu items
    fileMenu.setActionCommand("File");
    fileMenu.setMnemonic((int)'F');
    openQFDMenuItem.setActionCommand("Open QFD");
    openQFDMenuItem.setMnemonic((int)'O');
    openQFDMenuItem.addActionListener(this);

    fileMenu.add(openQFDMenuItem);
    saveQFDMenuItem.setActionCommand("Save QFD");
    saveQFDMenuItem.setMnemonic((int)'S');
    saveQFDMenuItem.addActionListener(this);
    fileMenu.add(saveQFDMenuItem);
    deleteQFDMenuItem.setActionCommand("Delete QFD");
    deleteQFDMenuItem.setMnemonic((int)'D');
    deleteQFDMenuItem.addActionListener(this);
    fileMenu.add(deleteQFDMenuItem);
    fileMenu.addSeparator();
    exitMenuItem.setActionCommand("Exit");
    exitMenuItem.setMnemonic((int)'X');
}

```

```

        exitMenuItem.addActionListener(this);
        fileMenu.add (exitMenuItem);

        // the view menu item with user-defined and trace view menu items
        viewMenu.setActionCommand("View");
        viewMenu.setMnemonic((int)'V');
        userDefinedViewMenuItem.setActionCommand("User-Defined");
        userDefinedViewMenuItem.setMnemonic((int) 'D');
        userDefinedViewMenuItem.addActionListener(this);
        viewMenu.add (userDefinedViewMenuItem);
        traceViewMenuItem.setActionCommand("Trace");
        traceViewMenuItem.setMnemonic((int) 'C');
        traceViewMenuItem.addActionListener(this);
        viewMenu.add(traceViewMenuItem);

        setJMenuBar(bar);
        bar.add(fileMenu);
        bar.add(viewMenu);

        // JTable configuration
        leftTable = new JTable(leftElements);
        leftTable.setPreferredScrollableViewportSize(new Dimension(500, 70));
        depTable = new JTable(dependencyElements);
        depTable.setPreferredScrollableViewportSize(new Dimension(500, 70));
        depTable.setBackground(Color.yellow);
        topTable = new JTable(topElements);
        topTable.setPreferredScrollableViewportSize(new Dimension(500, 70));

        // Create a scroll pane fore each JTable
        topScrollPane = new JScrollPane();
        depScrollPane = new JScrollPane();
        leftScrollPane = new JScrollPane();
        // Create a new table instance
        getContentPane().setLayout(new java.awt.GridBagLayout());
        getContentPane().add(calcButton, new
java.awt.GridBagConstraints(0,2,1,1,0,0,0,0,java.awt.GridBagConstraints.CENTER,java.awt.GridBagConstraints.NONE,new
java.awt.Insets(0,0,0,0),0,0));
        getContentPane().add(exitButton, new
java.awt.GridBagConstraints(0,1,1,1,0,0,0,0,java.awt.GridBagConstraints.CENTER,java.awt.GridBagConstraints.NONE,new
java.awt.Insets(0,0,0,0),0,0));
        getContentPane().add(depScrollPane,
        new java.awt.GridBagConstraints(1, 3, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.NONE,
        new java.awt.Insets(0, 0, 0, 0), 0, 0));
        getContentPane().add(leftScrollPane,
        new java.awt.GridBagConstraints(0, 3, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.NONE,
        new java.awt.Insets(0, 0, 0, 0), 0, 0));
        getContentPane().add(topScrollPane,
        new java.awt.GridBagConstraints(1, 2, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(0, 0, 3, 0), 0, 0));
        // Scroll configuration for left JTable
        leftScrollPane.setPreferredSize(new java.awt.Dimension(300,300));
        leftScrollPane.setSize(new java.awt.Dimension(300,300));
        leftScrollPane.setMinimumSize(new java.awt.Dimension(300, 300));

        leftScrollPane.setHorizontalScrollBarPolicy(javax.swing.JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        leftScrollPane.setVerticalScrollBarPolicy(javax.swing.JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

        // Scroll configuration for middle JTable
        depScrollPane.setMinimumSize(new java.awt.Dimension(500, 300));
        depScrollPane.setPreferredSize(new java.awt.Dimension(500, 300));
        depScrollPane.setViewportBorder(null);
        depScrollPane.setVerticalScrollBarPolicy(javax.swing.JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

        depScrollPane.setHorizontalScrollBarPolicy(javax.swing.JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        // Scroll configuration for top JTable
        topScrollPane.setMinimumSize(new java.awt.Dimension(300, 100));
        topScrollPane.setPreferredSize(new java.awt.Dimension(300, 100));
        topScrollPane.setSize(new java.awt.Dimension(300, 100));

```

```

        topScrollPane.setVerticalScrollBarPolicy(javax.swing.JScrollPane.VERTICAL_SCROLLBAR_NEVER);

        topScrollPane.setHorizontalScrollBarPolicy(javax.swing.JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        topScrollPane.getViewport().add(topTable);
        topTable.setBounds(new java.awt.Rectangle(21,25,50,50));
        topTable.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        topTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        depScrollPane.getViewport().add(depTable);

        // middle Jtable configuration
        depTable.setBounds(new java.awt.Rectangle(14,20,50,50));
        depTable.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        depTable.setCellSelectionEnabled(true);
        depTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);

        // left jTable configuration
        leftScrollPane.getViewport().add(leftTable);
        leftTable.setBounds(new java.awt.Rectangle(22,18,50,50));
        leftTable.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        leftTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_ALL_COLUMNS);
        setResizable(true);
        setSize(new java.awt.Dimension(650, 700));

        this.pack();

        this.setVisible(true);
        setBounds(new java.awt.Rectangle(0, 0, 822, 516));
    }
    /**
    * saveQFD: save this class to a file
    * @return void
    * @param void
    */
    public void saveQFD () {
        // create a SAVE FileDialog
        FileDialog fileDialog = new FileDialog(this,"Save", FileDialog.SAVE);
        fileDialog.setDirectory(projectName);
        fileDialog.show();

        // read the users file choice
        String dir = fileDialog.getDirectory();
        String file = fileDialog.getFile();
        // save file with .cfg extension
        if (file.indexOf("qfd") == -1) {
            if (file.indexOf(".") == -1)
                file = file + ".qfd";
            else
            {
                if (file.substring(file.length() -1, file.length()) == ".")
                    file = file + "qfd";
            }
        }

        try {
            FileOutputStream f = new FileOutputStream(dir + file);
            ObjectOutputStream s = new ObjectOutputStream(f);
            s.writeObject(this);
            s.flush();
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
    /**
    * openQFD: open a file of an existing QFD matrix of class HouseMatrix
    * @return void
    * @param void
    */
    public void openQFD () {
        // create a LOAD FileDialog
        FileDialog fileDialog = new FileDialog (this,"Open", FileDialog.LOAD);
        fileDialog.show();
    }

```

```

        // what file and from which directory did the user select?
        String dir = fileDialog.getDirectory();
        String file = fileDialog.getFile();
        try {
            FileInputStream f = new FileInputStream (dir + file);
            ObjectInputStream s = new ObjectInputStream(f);
            HouseMatrix house = (HouseMatrix)s.readObject();
            // switch to new house

            house.show();
            house.repaint();
            this.dispose();

        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
/**
 * this method performs operation (forward or reverse calc and save)
 * when user presses the calc button.
 */
public void calcButtonActionPerformed () {
    CalcMatrix cm = new CalcMatrix(leftElements, topElements, dependencyElements);
    Tools tempTools = new Tools();
    if (tempTools.StringToInt(this.leftID) >= tempTools.StringToInt(this.origin))
        cm.forwardCalculation();
    else
        cm.reverseCalculation();
    saveButtonActionPerformed();
}
}
/**
 * this method saves new information over old information
 */
public void saveButtonActionPerformed () {
    if (isHouse) {
        for (int i=0; i<this.oldTopElements.getRowCount(); i++)
            this.oldTopElements.setValueAt(topElements.getValueAt(2,i),i,2);
        this.saveHouse=true;
    }
    oldDepElements.setDataVector(dependencyElements.getDataVector(),dependencyElements.getColumnIdentifiers());
}
}
/**
 * Action event handler for the menu events.
 *
 * @param e The action event that is created by selecting
 * a menu item from this menu
 */
public void actionPerformed (ActionEvent event) {
    MatrixCalculator mc = new MatrixCalculator();
    Dialog dialog = new Dialog(this,true); // ensure the view closes before proceeding
    Object object = event.getSource();
    if (object == openQFDMenuItem) // user wants to open a QFD HOQ
        openQFD();
    else if (object == saveQFDMenuItem) // user wants to save a QFD HOQ
        saveQFD();
    else if (object == userDefinedViewMenuItem) { // user wants to conduct a user-defined view
        double nTimes; // a variable to store 'n' value
        boolean flagValue ; // a flag (true == > and false == < )
        UserDefinedViewGUI newView = new UserDefinedViewGUI(dialog);
        nTimes = Double.parseDouble(newView.getNValue());
        flagValue = newView.getIsGreaterThan();
        new UserDefinedView (projectName, houseVersion, leftElements, mc.transposeComponent(topElements),
        dependencyElements, nTimes, flagValue);
        newView.dispose();
    }
    else if (object == traceViewMenuItem) { // user wants to conduct a dependency trace view

```

```

        int colSelected;           // a variable to track row or col selected
        double viewValue;        // a variable to capture value to filter on
        // check if user wants to filter on a column or row
        Object[] options = { "Column", "Row" };
        int j = JOptionPane.showOptionDialog(this, "Would you like to trace on a Column or Row?",
        "Question", JOptionPane.DEFAULT_OPTION, JOptionPane.QUESTION_MESSAGE, null, options, options[0]);
        if( j==0 ){ // trace on a column
            TraceViewGUI traceView = new TraceViewGUI(dialog, topElements);
            viewValue = Double.parseDouble(traceView.getFilterValue());
            colSelected = traceView.getColumnSelected();
            new TraceUpstream (projectName, houseVersion, topID, depIndex, viewValue,
colSelected);
            new TraceDownstream (projectName, houseVersion, topID, numOfComponents,
depIndex, viewValue, colSelected);
        } else { // trace on a row
            TraceViewGUI traceView = new TraceViewGUI(dialog, leftElements, j);
            viewValue = Double.parseDouble(traceView.getFilterValue());
            colSelected = traceView.getColumnSelected();
            new TraceUpstream (projectName, houseVersion, leftID, depIndex, viewValue,
colSelected);
            new TraceDownstream (projectName, houseVersion, leftID, numOfComponents,
depIndex, viewValue, colSelected);
        }
    }

    else if (object == deleteQFDMenuItem) // user wants to delete a QFD HOQ
        System.out.println("Use DOS to delete QFD files...");
    else if (object == exitMenuItem) // user wants to close window
        this.dispose();
    else if (object == calcButton) { // user pressed calc button
        calcButtonActionPerformed();
    }
    else if (object == exitButton) { // user pressed save & exit button
        saveButtonActionPerformed ();
        this.dispose();
    }
}

/**
 * main: main procedure for class testing
 * @return void
 * @param String[] args -- but not used
 */
public static void main (String[] args) {
    Object[] names = {"", "", ""};
    Object[] depNames = {"", "", "", ""};
    Object value = new Integer(2);
    // QFD upstream example
    Object[][] tableData = {{ "R1", "good", "5"}, { "R2", "fast", "1"}, { "R3", "cheap", "3"} };
    Object[][] tableData2 = {{ "S1", "100%", "0.53"}, { "S2", "100mph", "1.06"}, { "S3", "$100.00", "4.76"}, { "S4", "100psi", "2.65"} };
    Object[][] tableDep = {{ "0", "1", "0", "3"}, { "3", "1", "0", "0"}, { "0", "0", "9", "0"} };

    String project = new String("MyProject");

    MyDefaultTableModel t1 = new MyDefaultTableModel (tableData,names);
    MyDefaultTableModel t2 = new MyDefaultTableModel (tableData2,names);
    t1.join(t2);
    MyDefaultTableModel matrix = new MyDefaultTableModel (tableDep, depNames);
    t1.setOrigin(true);
    HouseMatrix house = new HouseMatrix(project, "Risk", 1, -1, "aComponent", t1,t2, matrix, "aComponent", null, "0");
    house.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            System.exit(0);
        }
    } );
}
}

```


5. Cases.QFD.MyDefaultTableModel

```
/**-----
 * Filename: MyDefaultTableModel.java
 * @Date: 1-20-2003
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: This class is a specialization of the DefaultTableModel class.
 * It allows CASES to perform function not available to a default table model
 * (e.g. is Cell Editable based on a origin variable and make clone functionality)
 *-----
 */
package Cases.QFD;

import java.io.Serializable;
import java.util.Vector;
import javax.swing.table.DefaultTableModel;

/**
 * This class adds a method to retrieve the columnIdentifiers
 * which is needed to implement the removal of
 * column data from the table model and a method to not allow cell editing.
 * @testcase test.Cases.QFD.TestMyDefaultTableModel
 */
public class MyDefaultTableModel extends DefaultTableModel implements Serializable {
    /**
     * the default constructor -- overrides super constructor
     * constructs a default defaulttablemodel which is a table of zero columns
     * and zero rows.
     */
    public MyDefaultTableModel () {
        super();
    }

    /**
     * overrides super constructor
     * constructs a defaulttablemodel with as many columns
     * as there are elements in columnNames and rowCount of null object values.
     */
    public MyDefaultTableModel(Vector columnNames, int rowCount){
        super(columnNames, rowCount);
    }

    /**
     * overrides super constructor
     * constructs a defaulttablemodel with as many columns as
     * there are elements in columnNames and rowCount of null object values.
     */
    public MyDefaultTableModel(Object[] columnNames, int rowCount){
        super(columnNames, rowCount);
    }

    /**
     * overrides super constructor
     * constructs a defaulttablemodel and initializes the table by passing
     * data and columnNames to the super setDataVector method.
     */
    public MyDefaultTableModel(Object[][] data, Object[] columnNames){
        super(data, columnNames);
    }

    /**
     * overrides super constructor
     * constructs a DefaultTableModel with rows and cols of null object values.
     */
    public MyDefaultTableModel(int rows, int cols){
        super(rows, cols);
    }

    /**
     * returns the column identifiers in the model.

```

```

    * @return Vector
    */
    public Vector getColumnIdentifiers() {
        return this.columnIdentifiers;
    }
    /**
     * returns a clone of the table model
     * @return MyDefaultTableModel
     */
    public MyDefaultTableModel makeClone() {
        int row = this.getRowCount();
        int col = this.getColumnCount();
        MyDefaultTableModel tempTable = new MyDefaultTableModel(row,col);
        for (int i=0; i<row; i++)
            for (int j=0; j<col; j++)
                tempTable.setValueAt(this.getValueAt(i,j),i,j);
        tempTable.setColumnIdentifiers(this.getColumnIdentifiers());
        return tempTable;
    }
    /**
     * allows cell editing if it is the origin
     * returns the value of isOrigin.
     * @return boolean
     */
    public boolean isCellEditable(int row, int col) {
        return isOrigin();
    }
    /**
     * joins a table
     */
    public void join (MyDefaultTableModel passTable) {
        Vector tempVector = passTable.getDataVector();
        for (int i=0; i<tempVector.size(); i++) {
            // System.out.println(tempVector.get(i).toString());
            this.addRow((Vector)tempVector.get(i));
        }
    }
    /**
     * returns the value of origin variable
     * @return boolean
     */
    public boolean isOrigin(){ return origin; }
    /**
     * sets the value of origin variable
     * @param boolean
     */
    public void setOrigin(boolean origin){ this.origin = origin; }

    /** private variable for storing state of origin */
    private boolean origin;
}

```

6. Cases.QFD.StepQFD

```

/**-----
 * Filename: StepQFD.java
 * @Date: 4-26-2003
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: the aggregation of QFD information for Step class
 *-----
 */

```

```

package Cases.QFD;
import java.io.Serializable;

```

```

public class StepQFD implements Serializable {
    /** this table is used for steps is drawn direction and downstream bridge steps */
    MyDefaultTableModel stepTable;
    /** this table is a special table for upstream bridge steps */
    private MyDefaultTableModel upstreamStepTable;

    /** @return MyDefaultTableModel step table */
    public MyDefaultTableModel getStepTable() { return stepTable; }
    /** @param MyDefaultTableModel new step table */
    public void setStepTable(MyDefaultTableModel stepTable) { this.stepTable = stepTable; }

}

```

7. Cases.QFD. TraceDownstream

```

/**-----
 * Filename: TraceForward.java
 * @Date: 2-25-2003
 * @Author: Arthur Clomera for NPS Thesis
 * @Compiler: JDK 1.3.1
 * Description: This class recursively traces forward on a trace value and col selected.
 *-----
 */

package Cases.QFD;

import java.util.Vector;
import Cases.CasesTitle;
import Cases.ComponentType;
import Cases.GUI.util.Tools;
import Cases.ProjectSchemaFrame;
import Cases.QFD.util.AVCTool;
import Cases.StepType;

/**
 * This class recursively traces forward on a trace value and col selected.
 */
public class TraceDownstream implements CasesTitle {
    /**
     * constructor for TraceDownString based on project, currentVersion, and dep.
     * Requires information about startComponentID, endComponentID (to get range of trace).
     * Needs rowSelected and viewValue to trim view.
     * @param String, String, String, int, int, double, and int
     */
    public TraceDownstream(String project, String currentVersion, String startComponentID, int endComponent, int dep, double
viewValue, int rowSelected) {
        /** a vector for storing downstream components */
        Vector downstreamRowVector=new Vector();
        // mark the end point
        this.endPoint = endComponent;
        // store the project name
        this.projectName = project;
        // which dependency are we tracing on
        this.dependency = dep;
        // which value are we filtering on
        this.viewValue = viewValue;
        // the first row to trace on is the one provided by the user
        downstreamRowVector.add(new Integer(rowSelected));
        // call the method to start trace
        executeDownstreamTrace (currentVersion, startComponentID, downstreamRowVector);
    }

    /**
     * recursive call to trace on a versionString, downCompID, and rowVector components

```

```

    * @param String, String, and Vector
    */
    public void executeDownstreamTrace(String versionString, String downCompID, Vector rowVector) {
        Tools tempTool = new Tools();
        AVCTool tempAVCTool = new AVCTool (this.projectName);
        String traceStepID;
        TraceDownstreamView tempView;
        // variables to hold QFD information
        MyDefaultTableModel leftTable;
        MyDefaultTableModel topTable;
        MyDefaultTableModel matrix;
        // project schema is based on project name
        this.psf = new ProjectSchemaFrame(this.projectName);
        Vector tempDownVector = tempAVCTool.getSecondaryOutputComponents(downCompID);
        // parse through tempDownVector to perform trace on each component in vector
        for (int downLoop = 0; downLoop < tempDownVector.size(); downLoop++) {
            if ((tempTool.StringToInt(downCompID) != this.endPoint-1) && (rowVector.size() > 0)) {
                // construct stepID from current component ID and secondary output component ID
                traceStepID = "s-
"+tempTool.intToString(tempTool.StringToInt(downCompID))+tempDownVector.get(downLoop);
                // get QFD information based on traceStepID
                leftTable = (MyDefaultTableModel)
                    ((ComponentQFD)
                    ((ComponentType)psf.compHashtable.get(
downCompID)).getComponentHashtable().get(versionString)).getComponentTable(dependency);
                    topTable = ((ComponentQFD)
                    ((ComponentType)psf.compHashtable.get( tempDownVector.get(downLoop)
)).getComponentHashtable().get(versionString)).getComponentTable(dependency);
                    matrix = ((StepQFD)
                    ((StepType)psf.stepHashtable.get(traceStepID)).getStepQFDHashtable().get(versionString)).getStepTable();
                    tempView = new TraceDownstreamView (this.projectName, versionString, leftTable,
topTable, matrix, viewValue, rowVector);
                    // recursive step
                    executeDownstreamTrace (versionString, (String)tempDownVector.get(downLoop),
tempView.getColVector());
                }
            }
        }
        /** variable to store end point */
        private int endPoint;
        /** variable to store dependency */
        private int dependency;
        /** variable to point to PSF */
        private ProjectSchemaFrame psf;
        /** string to store the project name */
        private String projectName;
        /** double value to hold the viewValue */
        private double viewValue;
    }
}

```

8. Cases.QFD. TraceDownstreamView

```

/**
-----
* Filename: TraceDownStreamView.java
* @Date: 1-31-2003
* @Author: Arthur Clomera for NPS Thesis
* @Compiler: JDK 1.3.1
* Description: This class modifies tables to view in HouseMatrix.
-----
*/

package Cases.QFD;

import java.io.Serializable;
import java.util.Vector;

```

```

import Cases.QFD.util.GetDoubleValue;

/**
 * This class modifies tables to view in HouseMatrix.
 */
public class TraceDownstreamView implements Serializable {

    /**
     * This constructor will do a downstream trace based on a project and currentVersion.
     * It requires the 3 table models to construct the HOQ.
     * It uses the rowVector and filterValue to trim view.
     * @param String, String, MyDefaultTableModel (x3), double, and Vector.
     */
    public TraceDownstreamView(String project, String currentVersion, MyDefaultTableModel left, MyDefaultTableModel top,
        MyDefaultTableModel dep, double filterValue, Vector rowVector) {
        Object[] names = {"", ""};
        colVector = new Vector();
        // initialize temporary table models to match row vector and no top rows yet. Top rows will be added later.
        MyDefaultTableModel tempLeft = new MyDefaultTableModel(names, rowVector.size()); // initialize to size of row vector
        MyDefaultTableModel tempTop = new MyDefaultTableModel(names, 0);
        MyDefaultTableModel tempDep = new MyDefaultTableModel(dep.getColumnIdentifiers(), rowVector.size());
        // populate the left matrix to match rowVector
        for (int row=0; row < rowVector.size(); row++){
            tempLeft.setValueAt(left.getValueAt(((Integer)rowVector.get(row)).intValue(), 0), row, 0);
            tempLeft.setValueAt(left.getValueAt(((Integer)rowVector.get(row)).intValue(), 1), row, 1);
        }
        // an object to perform double value operations
        GetDoubleValue gdv = new GetDoubleValue();
        this.colVector.removeAllElements();
        int depCol=0;
        boolean depFlag = false; // flag to indicate a dependency has been added
        boolean testFlag = false; // tests if a row needs to be added to the top matrix
        for (int i=0; i<dep.getColumnCount(); i++) {
            for (int loop=0; loop<rowVector.size(); loop++) {
                // check if value meets the filter value
                if (gdv.getValue(dep.getValueAt(((Integer)rowVector.get(loop)).intValue(), i)) >= filterValue) {
                    for (int testLoop=0; testLoop<this.colVector.size(); testLoop++) // loop through col vector
                        if (((Integer)this.colVector.get(testLoop)).intValue() == i) // check that it hasn't been added already
                            testFlag = true;
                    // must have been added
                    if (!testFlag) { // add a row to the top matrix along with it's values
                        tempTop.addRow(names);
                        tempTop.setValueAt(top.getValueAt(i, 0), tempTop.getRowCount()-1, 0);
                        tempTop.setValueAt(top.getValueAt(i, 1), tempTop.getRowCount()-1, 1);
                        this.colVector.add(new Integer(i));
                    }
                    depFlag=true;
                    tempDep.setValueAt(dep.getValueAt(((Integer)rowVector.get(loop)).intValue(), i), loop, depCol);
                }
                testFlag = false;
            }
            if (depFlag) { // must have added a dependency in middle matrix
                depFlag = false;
                depCol++;
            }
        }
        // set the column and row counts so data will not be zeroed out by HouseMatrix
        tempDep.setColumnCount(tempTop.getRowCount());
        tempDep.setRowCount(tempLeft.getRowCount());
        String windowLabel = "Downstream Component Trace View (" + currentVersion + ")";
        if ((tempLeft.getRowCount() >= 1) && (tempTop.getRowCount() >= 1)) // show tempView if left or top have at least one row
            each
                this.setHouse(new HouseMatrix (project, windowLabel, -1, -1, null, tempLeft, tempTop, tempDep, null, null,
                    currentVersion));
        }
    }

```

```

    /**
    * get House : returns house variable
    * @return HouseMatrix
    */
    public HouseMatrix getHouse(){ return house; }

    /**
    * set House : sets house variable
    * @param HouseMatrix
    */
    public void setHouse(HouseMatrix house){ this.house = house; }

    /**
    * returns the variable colVector
    * @return Vector
    */
    public Vector getColVector(){ return colVector; }
    /** HouseMatrix for holding table models */
    private HouseMatrix house;
    /** Vector variable to store columns captured by trace */
    private Vector colVector;
}

```

9. Cases.QFD. TraceUpstream

```

/**-----
* Filename: TraceUpstream.java
* @Date: 2-25-2003
* @Author: Arthur Clomera for NPS Thesis
* @Compiler: JDK 1.3.1
* Description: This class recusively traces backward on a trace value and col selected.
*-----
*/

package Cases.QFD;

import java.util.Vector;
import Cases.CasesTitle;
import Cases.ComponentType;
import Cases.GUI.util.Tools;
import Cases.ProjectSchemaFrame;
import Cases.QFD.util.AVCTool;
import Cases.StepType;

/**
* TraceUpstream : traces components upstream with a specified filter value
*/
public class TraceUpstream implements CasesTitle {
    /**
    * Constructor for upstream trace based on project, dep, and currentVersion
    * requires startID and trims view based on viewValue and colSelected.
    * @param String, String, String, int, double, and int.
    */
    public TraceUpstream(String project, String currentVersion, String startID, int dep, double viewValue, int colSelected) {
        Tools newTool = new Tools();
        /** a vector for storing upstream components */
        Vector upstreamColVector=new Vector();
        // store the project name
        this.projectName = project;
        // which dependency to trace on
        this.dependency = dep;
        // which value to filter on
        this.viewValue = viewValue;
        // the first column to trace on will be provided by the user
        upstreamColVector.add(new Integer(colSelected));
        // call the method to start the trace
        executeUpstreamTrace (currentVersion, startID, upstreamColVector);
    }
}

```

```

    }
    /**
    * recursive call to trace on a versionString, upComp, and colVector components
    * @param String, String, and Vector
    */
    public void executeUpstreamTrace(String versionString, String upComp, Vector colVector) {
        // project schema is based on project name
        this.psf = new ProjectSchemaFrame(this.projectName);
        Tools xTool = new Tools();
        AVCTool tempAVCTool = new AVCTool(this.projectName);
        Vector tempUpVector = tempAVCTool.getSecondaryInputComponents(upComp);
        TraceUpstreamView tempView;

        String stepID;
        // variables to hold QFD information
        MyDefaultTableModel leftTable;
        MyDefaultTableModel topTable;
        MyDefaultTableModel matrix;

        // parse through tempUpVector to perform trace on each component in vector
        for (int upLoop=0; upLoop<tempUpVector.size(); upLoop++) {
            if ((xTool.StringToInt(upComp) != 0) && (colVector.size() > 0)) {
                stepID = "s-
"+xTool.intToString(xTool.StringToInt(((String)tempUpVector.get(upLoop))))+upComp;
                // get QFD information based on the stepID
                leftTable = ((ComponentQFD)
                    ((ComponentType)psf.compHashtable.get( tempUpVector.get(upLoop)
)).getComponentHashtable().get(versionString)).getComponentTable(dependency);
                topTable = ((ComponentQFD)
                    ((ComponentType)psf.compHashtable.get( upComp
)).getComponentHashtable().get(versionString)).getComponentTable(dependency);
                matrix = ((StepQFD)
                    ((StepType)psf.stepHashtable.get(stepID)).getStepQFDHashtable().get(versionString)).getStepTable();
                tempView = new TraceUpstreamView (this.projectName, versionString, leftTable,
topTable, matrix, viewValue, colVector);
                // recursive step

                executeUpstreamTrace(versionString,(String)tempUpVector.get(upLoop),tempView.getRowVector());
            }
        }
    }

    /** variable to store dependency */
    private int dependency;
    /** variable to point to PSF */
    private ProjectSchemaFrame psf;
    /** string to store the project name */
    private String projectName;
    /** double value to hold the viewValue */
    private double viewValue;
}

```

10. Cases.QFD.TraceUpstreamView

```

/**-----
* Filename: TraceUpstreamView.java
* @Date: 1-31-2003
* @Author: Arthur Clomera for NPS Thesis
* @Compiler: JDK 1.3.1
* Description: This class modifies tables to view in HouseMatrix.
*-----
*/

package Cases.QFD;

import java.io.Serializable;
import java.util.Vector;

```

```

import Cases.QFD.util.GetDoubleValue;

/**
 * This class modifies tables to view in HouseMatrix.
 */
public class TraceUpstreamView implements Serializable {

    /**
     * This constructor will do an upstream trace based on a project and currentVersion.
     * It requires the 3 table models to construct the HOQ.
     * It uses the colVector and filterValue to trim view.
     * @param String, String, MyDefaultTableModel (x3), double, and Vector.
     */
    public TraceUpstreamView(String project, String currentVersion, MyDefaultTableModel left, MyDefaultTableModel top,
        MyDefaultTableModel dep, double filterValue, Vector colVector) {
        Object[] names = {"", ""};
        rowVector = new Vector();
        // initialize temporary table models to match row vector and no left rows yet. Left rows will be added later.
        MyDefaultTableModel tempLeft = new MyDefaultTableModel(names, 0);
        MyDefaultTableModel tempTop = new MyDefaultTableModel(names, colVector.size()); // initialize colVector
        MyDefaultTableModel tempDep = new MyDefaultTableModel(dep.getColumnIdentifiers(), left.getRowCount());
        // populate the top matrix to match colVector
        for (int col=0; col < colVector.size(); col++){
            tempTop.setValueAt(top.getValueAt(((Integer)colVector.get(col)).intValue(), 0), col, 0);
            tempTop.setValueAt(top.getValueAt(((Integer)colVector.get(col)).intValue(), 1), col, 1);
        }
        // an object to perform double value operations
        GetDoubleValue gdv = new GetDoubleValue();
        this.rowVector.removeAllElements();
        int depRow=0;
        boolean depFlag = false; // flag to indicate a dependency has been added
        boolean testFlag = false; // tests if a row needs to be added to the top matrix
        for (int i=0; i<dep.getRowCount(); i++) {
            for (int loop=0; loop<colVector.size(); loop++) {
                // check if value meets the filter value
                if (gdv.getValue(dep.getValueAt(i, ((Integer)colVector.get(loop)).intValue())) >= filterValue) {
                    for (int testLoop=0; testLoop<this.rowVector.size(); testLoop++) // loop through row vector
                        if (((Integer)this.rowVector.get(testLoop)).intValue() == i) // check that it hasn't been added
                            already
                                testFlag = true;
                                // must have been added
                                if (!testFlag) { // add a row to the left matrix along with it's values
                                    tempLeft.addRow(names);
                                    tempLeft.setValueAt(left.getValueAt(i, 0), tempLeft.getRowCount()-1, 0);
                                    tempLeft.setValueAt(left.getValueAt(i, 1), tempLeft.getRowCount()-1, 1);
                                    this.rowVector.add(new Integer(i));
                                }
                                tempDep.setValueAt(dep.getValueAt(i, ((Integer)colVector.get(loop)).intValue()), depRow, loop);
                                depFlag=true;
                            }
                            testFlag = false;
                        }
                    }
                if (depFlag) { // must have added a dependency in middle matrix
                    depFlag = false;
                    depRow++;
                }
            }
        }

        // set the column and row counts so data will not be zeroed out by HouseMatrix
        tempDep.setColumnCount(tempTop.getRowCount());
        tempDep.setRowCount(tempLeft.getRowCount());
        String windowLabel = "Upstream Component Trace View (" + currentVersion + ")";
        if ((tempTop.getRowCount() >= 1) && (tempLeft.getRowCount() >= 1)) // show tempView if left or top have at least one row
            each
                this.setHouse(new HouseMatrix (project, windowLabel, -1, -1, null, tempLeft, tempTop, tempDep, null, null,
                    currentVersion));
    }
}

```



```

    }
    /**
    * get row vector
    * @return Vector
    */
    public Vector getRowVector(){ return rowVector; }
    /**
    * get House : returns house variable
    * @return HouseMatrix
    */
    public HouseMatrix getHouse(){ return house; }
    /**
    * set House : sets house variable
    * @param HouseMatrix
    */
    public void setHouse(HouseMatrix house){ this.house = house; }
    /** Vector variable to store rows captured by trace */
    private Vector rowVector;
    /** HouseMatrix for holding table models */
    private HouseMatrix house;
}

```

11. Cases.QFD.TraceViewGUI

```

/**-----
 * @Filename: TraceViewGUI.java
 * @Date: 1-31-2003
 * @Author: Arthur Clomera for NPS Thesis
 * @Compiler: JDK 1.3.1
 * Description: This is a dialog window to get information to display a
 * Trace View.
 *-----
 */

package Cases.QFD;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

/**
 * This is a dialog window to get information to display a Trace View.
 */
public class TraceViewGUI extends JDialog {
    /** Default constructor for testing */
    public TraceViewGUI() {
        initGUI();
        for (int i=0; i<10; i++) {
            Integer tempInteger = new Integer(i+1);
            this.columnChoice.add(tempInteger.toString());
        }
        this.rowFlag=false;
        setSize(415,300);
        setVisible(true);
    }
    /** Polymorphic constructor to activate dialog */
    public TraceViewGUI (Dialog owner, MyDefaultTableModel top) {
        super (owner,"Trace View Parameters",true);
        this.rowFlag = false;
        initGUI();
        for (int i=0; i<top.getColumnCount(); i++) {
            Integer tempInteger = new Integer(i+1);
            String temp = " " + top.getValueAt(0,i) + " ";

```

```

        this.columnChoice.add(temp);
    }
    setSize(415,300);
    setVisible(true);
}
/** Polymorphic constructor to activate dialog */
public TraceViewGUI (Dialog owner, MyDefaultTableModel left, int rowFlag) {
    super (owner,"Trace View Paramenters",true);
    this.rowFlag = true;
    initGUI();
    for (int i=0; i<left.getRowCount(); i++) {
        Integer tempInteger = new Integer(i+1);
        String temp = " " + left.getValueAt(i,0) + " ";
        this.columnChoice.add(temp);
    }
    setSize(415,300);
    setVisible(true);
}
/** initGUI() is the main method for displaying initializing the GUI */
public void initGUI() {
    okayButton.setLabel("OK");
    okayButton.setActionCommand("okayButton");
    filterTextField.setText("");
    getContentPane().setLayout(new java.awt.GridBagLayout());
    // change the text based on row flag
    if (rowFlag) // print select row in choice label
        columnChoice.add("Select Row");
    else // print select column in choice label
        columnChoice.add("Select Column");
    // GUI component configuration
    getContentPane().add(columnChoice,
        new java.awt.GridBagConstraints(1, 0, 2, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(2, 0, 7, 0), 0, 0));
    getContentPane().add(label2,
        new java.awt.GridBagConstraints(0, 1, 2, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(0, 0, 0, 33), 0, 0));
    getContentPane().add(label1,
        new java.awt.GridBagConstraints(0, 0, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.NONE,
        new java.awt.Insets(0, 0, 2, 46), -22, 0));
    getContentPane().add(filterTextField,
        new java.awt.GridBagConstraints(2, 1, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(1, 0, 3, 0), 0, 0));
    getContentPane().add(okayButton, new
java.awt.GridBagConstraints(0,3,1,1,0.0,0.0,java.awt.GridBagConstraints.CENTER,java.awt.GridBagConstraints.NONE,new
java.awt.Insets(0,0,0,0),0,0));
    setTitle("Trace View");
    setBounds(new java.awt.Rectangle(0,0,340,153));
    // change the message based on rowflag
    if (rowFlag) // user must want to trace from a row
        label1.setText(" Which row location to trace from : ");
    else // user must want to trace from a column
        label1.setText(" Which column location to trace from : ");
    label2.setText("Enter Filter Value : ");
    okayButton.addActionListener(new ActionListener(){public void actionPerformed(ActionEvent e){okayButtonActionPerformed
(e);}});
}
/** main procedure for unit testing. */
public static void main (String[] args)
{
    TraceViewGUI application = new TraceViewGUI();
    application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
/**
 * gets the filterValue

```

```

    * @return String
    */
    public String getFilterValue () {
        return this.filterTextField.getText();
    }
    /**
    * gets the column selected
    * @return int
    */
    public int getColumnSelected () {
        return this.columnChoice.getSelectedIndex()-1;
    }
    /**
    * this is the procedure which handles okay button actions
    */
    public void okayButtonActionPerformed(ActionEvent e) {
        double amount;
        // user must enter a numeric value to trace on
        try {
            amount = Double.valueOf(getFilterValue()).doubleValue();
        } catch (java.lang.NumberFormatException e2) {
            JOptionPane.showMessageDialog(this,"You must enter a numeric value to filter on","Invalid
Input",JOptionPane.ERROR_MESSAGE);
            return;
        }
        // user must select a column past the title in the column combobox
        if (getColumnSelected()<0)
            JOptionPane.showMessageDialog(this,"You must select a column","Invalid Selection",JOptionPane.ERROR_MESSAGE);
        else
            this.hide();
    }
    /** label for gui */
    private Label label1 = new Label();
    /** yet another gui label */
    private Label label2 = new Label();
    /** textfield variable for filter text field */
    private TextField filterTextField = new TextField();
    /** choice between a column or row */
    private Choice columnChoice = new Choice();
    /** an okay button */
    private Button okayButton = new Button();
    /** a boolean flag : true for row false for column */
    private boolean rowFlag;
}

```

12. Cases.QFD.UpdateCombinedOrigin

```

/**-----
 * Filename: UpdateOrigin.java
 * @Date: 2-20-2003
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: This class synchronized QFD matrices
 * calculations (upstream or downstream) dependent based upon origin.
 *-----
 */

package Cases.QFD;

import java.util.Vector;
import Cases.CasesTitle;
import Cases.ComponentType;
import Cases.GUI.util.Tools;
import Cases.ProjectSchemaFrame;
import Cases.QFD.util.AVCTool;
import Cases.StepType;

```

```

/**
 * This object assumes that there is a start node at '0' (placed
 * first when drawing project schema)
 * and the last node is the highest value (placed last when
 * drawing project schema).
 */
public class UpdateCombinedOrigin implements CasesTitle {
    /**
     * constructor which updates combined house of quality origin based on currentVersion
     * compNumber, origin, and dep.
     * needs PSF to get other QFD information.
     */
    public UpdateCombinedOrigin(ProjectSchemaFrame newPSF, String currentVersion, int compNumber, String origin, int dep) {
        Tools tempTool = new Tools();
        this.origin = origin; // store origin component id
        this.dependency = dep; // store dependency
        this.updateVersion = currentVersion; // store current version
        this.psf = newPSF; // store link to project schema frame object
        this.projectName = psf.projectName; // store project name
        psf.compSaveButton_actionPerformed(null); // save component information prior to this operation
        this.numOfComponents = compNumber; // store the number of components in graph pane
    }

    /**
     * perform all upstream calculations base upon upComp component id
     * @param String
     */
    public void updateUpStreamMatrix(String upComp) {
        Tools tempTool = new Tools();
        AVCTool tempAVCTool = new AVCTool(this.projectName);
        /** a vector to store all component upstream */
        Vector tempUpVector = tempAVCTool.getSecondaryInputComponents(upComp);
        /** vector for handling all secondary input components */
        Vector leftVector = new Vector();
        /** vector for handling all correlation matrices for each component above */
        Vector matrixVector = new Vector();
        /** top table is based on upComp */
        MyDefaultTableModel topTable = ((ComponentQFD)
            ((ComponentType)psf.compHashtable.get( upComp
        )),getComponentHashtable().get(updateVersion)).getComponentTable(dependency);
        // loop through upstream vector
        for (int upLoop = 0; upLoop<tempUpVector.size(); upLoop++) {
            if ( tempTool.StringToInt(upComp) != 0 ) { // if the upComp isn't the start node do the following
                System.out.print(upComp+":UpdateOrigin:2ndary inputs: ");
                System.out.println(tempUpVector.get(upLoop));
                String stepID = "s-
                "+tempTool.intToString(tempTool.StringToInt(((String)tempUpVector.get(upLoop))))+upComp;
                // add the left table to the vector based on the stepID
                leftVector.add( ((ComponentQFD)
                    ((ComponentType)psf.compHashtable.get(
                tempUpVector.get(upLoop))).getComponentHashtable().get(updateVersion)).getComponentTable(dependency));
                // add the middle table to the vector based on the stepID
                matrixVector.add( ((StepQFD)
                    ((StepType)psf.stepHashtable.get(stepID)).getStepQFDHashtable().get(updateVersion)).getStepTable());
            }
        }
        // display the HOQ
        CombinedHouseMatrix tempHouse = new CombinedHouseMatrix (projectName, "temp house", dependency,
        this.numOfComponents, this.origin, leftVector, topTable, matrixVector, tempUpVector, upComp, updateVersion);
        // no need to show it because we are using it to perform updates
        tempHouse.hide();
        // System.out.println("UpdateOrigin:stepID"+stepID);
        // perform calculations
        tempHouse.calcButtonActionPerformed();
        // save to capture the changes
    }
}

```

```

        tempHouse.saveButtonActionPerformed();
    }
    /**
     * perform all downstream calculations based on the downComp component id
     * @param String
     */
    public void updateDownStreamMatrix(String downComp) {
        Tools tempTool = new Tools();
        AVCTool tempAVCTool = new AVCTool(this.projectName);
        /** a vector to store all component downstream */
        Vector tempDownVector = tempAVCTool.getSecondaryOutputComponents(downComp);
        // loop through upstream vector
        for (int downLoop = 0; downLoop < tempDownVector.size(); downLoop++) {
            if (tempTool.StringToInt(downComp) != this.numOfComponents-1) { // if the downComp isn't the end node do the following...
                System.out.print(downComp + ":UpdateOrigin:2ndary outputs: ");
                System.out.println(tempDownVector.get(downLoop));
                String stepID = "s-
"+tempTool.intToString(tempTool.StringToInt(downComp))+tempDownVector.get(downLoop);
                // get qfd data (left, top, and middle) base upon stepID
                MyDefaultTableModel leftTable = ((ComponentQFD)
                ((ComponentType)psf.compHashtable.get(
downComp)).getComponentHashtable().get(updateVersion)).getComponentTable(dependency);
                MyDefaultTableModel topTable = ((ComponentQFD)
                ((ComponentType)psf.compHashtable.get( tempDownVector.get(downLoop)
)).getComponentHashtable().get(updateVersion)).getComponentTable(dependency);
                MyDefaultTableModel matrix = ((StepQFD)
                ((StepType)psf.stepHashtable.get(stepID)).getStepQFDHashtable().get(updateVersion)).getStepTable());
                // get qfd data (left, top, and middle) base upon stepID
                HouseMatrix tempHouse = new HouseMatrix (projectName, "temp house", dependency, this.numOfComponents,this.origin,
leftTable, topTable, matrix, downComp, (String)tempDownVector.get(downLoop), updateVersion);
                // no need to display it because we are doing calculations only, therefore hide from user
                tempHouse.hide();
                System.out.println("UpdateOrigin:stepID:"+stepID);
                // perform calculations
                tempHouse.calcButtonActionPerformed();
                // save inorder to capture changes
                tempHouse.saveButtonActionPerformed();
            }
        }
    }
    /** track the origin */
    private String origin;
    /** a calcmatrix object to perform matrix calculations */
    private CalcMatrix calcMatrix;
    /** track the dependency number */
    private int dependency;
    /** a project schema frame to retrieve component and step information */
    private ProjectSchemaFrame psf;
    /** the project name */
    private String projectName;
    /** string for the version being updated */
    private String updateVersion;
    /** the number of components */
    private int numOfComponents;
}

```

13. Cases.QFD.UpdateOrigin

```

/**-----
 * Filename: UpdateOrigin.java
 * @Date: 2-20-2003
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: This class synchronized QFD matrices
 *              calculations (upstream or downstream) dependent upon origin.

```

```

*-----
*/

package Cases.QFD;

import java.util.Vector;
import Cases.CasesTitle;
import Cases.ComponentType;
import Cases.GUI.util.Tools;
import Cases.ProjectSchemaFrame;
import Cases.QFD.util.AVCTool;
import Cases.StepType;

/**
 * This object assumes that there is a start node at '0' (placed
 * first when drawing project schema)
 * and the last node is the highest value (placed last when
 * drawing project schema).
 */
public class UpdateOrigin implements CasesTitle {
    public UpdateOrigin(ProjectSchemaFrame newPSF, String currentVersion, int compNumber, String origin, int dep) {
        Tools tempTool = new Tools();
        this.origin = origin; // store origin component id
        this.dependency = dep; // store dependency
        this.updateVersion = currentVersion; // store current version
        this.psf = newPSF; // store link to project schema frame object
        this.projectName = psf.projectName; // store project name
        psf.compSaveButton_actionPerformed(null); // save component information prior to this operation
        this.numOfComponents = compNumber; // store the number of components in graph pane
    }

    /**
     * perform all upstream calculations based upon upComp component id
     * @param String
     */
    public void updateUpStreamMatrix(String upComp) {
        Tools tempTool = new Tools();
        AVCTool tempAVCTool = new AVCTool(this.projectName);
        /** a vector to store all component upstream */
        Vector tempUpVector = tempAVCTool.getSecondaryInputComponents(upComp);
        // loop through upstream vector
        for (int upLoop = 0; upLoop < tempUpVector.size(); upLoop++) {
            if (tempTool.StringToInt(upComp) != 0) { // if upComp isn't the start node do the following...
                // System.out.print(upComp+":UpdateOrigin:2ndary inputs: ");
                System.out.println(tempUpVector.get(upLoop));
                String stepID = "s-
"+tempTool.intToString(tempTool.StringToInt(((String)tempUpVector.get(upLoop))))+upComp;
                // get qfd table data based upon stepID
                MyDefaultTableModel leftTable = ((ComponentQFD)
                    ((ComponentType)psf.compHashtable.get(
tempUpVector.get(upLoop))).getComponentHashtable().get(updateVersion)).getComponentTable(dependency);
                MyDefaultTableModel topTable = ((ComponentQFD)
                    ((ComponentType)psf.compHashtable.get( upComp
)).getComponentHashtable().get(updateVersion)).getComponentTable(dependency);
                MyDefaultTableModel matrix = ((StepQFD)
                    ((StepType)psf.stepHashtable.get(stepID)).getStepQFDHashtable().get(updateVersion)).getStepTable());
                // create an instance of the HOQ
                HouseMatrix tempHouse = new HouseMatrix (projectName, "temp house", dependency, this.numOfComponents, this.origin,
                    leftTable, topTable, matrix, (String) tempUpVector.get(upLoop),
                    upComp, updateVersion);
                // hide the calculations from the user
                tempHouse.hide();
                // System.out.println("UpdateOrigin:stepID"+stepID);
                // perform the calculations
                tempHouse.calcButtonActionPerformed();
                // save any changes that calculations might have caused
                tempHouse.saveButtonActionPerformed();
            }
        }
    }
}

```

```

        //recursively call the next component
        updateUpStreamMatrix ((String)tempUpVector.get(upLoop));
    }
}
}
/** perform all downstream calculations */
public void updateDownStreamMatrix(String downComp) {
    Tools tempTool = new Tools();
    AVCTool tempAVCTool = new AVCTool(this.projectName);
    Vector tempDownVector = tempAVCTool.getSecondaryOutputComponents(downComp);
    for (int downLoop = 0; downLoop<tempDownVector.size(); downLoop++) {
        if (tempTool.StringToInt(downComp) != this.numOfComponents-1) {
            //
            System.out.print(downComp + ":UpdateOrigin:2ndary outputs: ");
            System.out.println(tempDownVector.get(downLoop));
            String stepID = "s-
            "+tempTool.intToString(tempTool.StringToInt(downComp))+tempDownVector.get(downLoop);
            // get qfd data (left, top, and middle) base upon stepID
            MyDefaultTableModel leftTable = ((ComponentQFD)
            ((ComponentType)psf.compHashtable.get(
            downComp)).getComponentHashtable().get(updateVersion)).getComponentTable(dependency);
            MyDefaultTableModel topTable = ((ComponentQFD)
            ((ComponentType)psf.compHashtable.get( tempDownVector.get(downLoop)
            )),getComponentHashtable().get(updateVersion)).getComponentTable(dependency);
            MyDefaultTableModel matrix = ((StepQFD)
            ((StepType)psf.stepHashtable.get(stepID)).getStepQFDHashtable().get(updateVersion)).getStepTable();
            // create instance of HOQ
            HouseMatrix tempHouse = new HouseMatrix (projectName, "temp house", dependency, this.numOfComponents,this.origin,
            leftTable, topTable, matrix,
            downComp,
            (String)tempDownVector.get(downLoop), updateVersion);
            // hide calculations from the user
            tempHouse.hide();
            //
            System.out.println("UpdateOrigin:stepID:"+stepID);
            // perform calculations
            tempHouse.calcButtonActionPerformed();
            // save table data to capture calculations
            tempHouse.saveButtonActionPerformed();
            // recursively call method to continue operation for all components
            updateDownStreamMatrix((String)tempDownVector.get(downLoop));
        }
    }
}
}
}
/** track the origin */
private String origin;
/** a calcmatrix object to perform matrix calculations */
private CalcMatrix calcMatrix;
/** track the dependency number */
private int dependency;
/** a project schema frame to retrieve component and step information */
private ProjectSchemaFrame psf;
/** the project name */
private String projectName;
/** a string to track version being updated */
private String updateVersion;
/** the number of components */
private int numOfComponents;
}

```

14. Cases.QFD.UserDefinedView

```

/**-----
* Filename: UserDefinedView.java
* @Date: 1-21-2003
* @Author: Arthur Clomera for NPS Thesis
* @Compiler: JDK 1.3.1

```

```

* Description: This class modifies tables to view in HouseMatrix.
* Allows the user to trim the view based upon  $t = \text{mean} \pm n \cdot \text{std}$ 
* where t is the dependency threshold value specified against
* the mean and standard deviation of each group of components and
* 'n' is a user specified scalar.
* The mean and std are based upon each different set of components
*-----
*/

package Cases.QFD;

import java.io.Serializable;
import java.util.Enumeration;
import java.util.Vector;
import javax.swing.*;
import javax.swing.table.TableColumn;
import Cases.QFD.util.GetDoubleValue;

/**
 * This class modifies tables to view in HouseMatrix.
 */
public class UserDefinedView implements Serializable {
    /**
     * the constructor for UserDefinedView for a project, dep, and currentVersion.
     * Requires the table models to construct the HOQ.
     * nTimes and greatFlag are used to filter view.
     * @param String, String, MyDefaultTableModel (x3), double, and boolean.
     */
    public UserDefinedView(String project, String currentVersion,
                           MyDefaultTableModel left, MyDefaultTableModel top, MyDefaultTableModel dep,
                           double nTimes, boolean greaterFlag) {
        Object[] names = {"", "", ""};
        Vector depNames = new Vector();
        // set up empty left and top matrices
        MyDefaultTableModel tempLeft = new MyDefaultTableModel(names, 0);
        MyDefaultTableModel tempTop = new MyDefaultTableModel(names, 0);
        // initialize vector depNames
        for (int i = 0; i < dep.getColumnCount(); i++)
            depNames.addElement("");
        // set up an empty middle matrix
        MyDefaultTableModel tempDep = new MyDefaultTableModel(depNames, dep.getRowCount());
        // create a JTable for middle matrix
        JTable table = new JTable(tempDep);
        // do not allow user to edit the table
        table.setModel(tempDep);
        double nValue = nTimes; // store the 'n' value from user
        double leftSum = 0.0, topSum = 0.0; // initialize sum values
        double leftSTD = 0.0, topSTD = 0.0; // initialize std values
        double leftMean = 0.0, topMean = 0.0; // initialize mean values
        /** an object to conduct double values */
        GetDoubleValue gdv = new GetDoubleValue();
        // get the left mean
        for (int i = 0; i < left.getRowCount(); i++)
            leftSum += gdv.getValue(left.getValueAt(i, 2));
        leftMean = leftSum / left.getRowCount();
        // get the left std
        for (int i = 0; i < left.getRowCount(); i++)
            leftSTD += gdv.square(gdv.getValue(left.getValueAt(i, 2))) - gdv.square(leftMean);
        leftSTD /= left.getRowCount() - 1;
        leftSTD = java.lang.Math.sqrt(leftSTD);
        // get the top mean
        for (int i = 0; i < top.getRowCount(); i++)
            topSum += gdv.getValue(top.getValueAt(i, 2));
        topMean = topSum / top.getRowCount();
        // get the top STD
        for (int i = 0; i < top.getRowCount(); i++)
            topSTD += gdv.square(gdv.getValue(top.getValueAt(i, 2))) - gdv.square(topMean);
        topSTD /= top.getRowCount() - 1;
    }
}

```



```

topSTD = java.lang.Math.sqrt(topSTD);
double leftDecisionValue = leftMean+(nValue*leftSTD);
double topDecisionValue = topMean+(nValue*topSTD);
// populate tempDep
for (int row=0; row<dep.getRowCount(); row++)
    for (int col=0; col<dep.getColumnCount(); col++)
        tempDep.setValueAt(dep.getValueAt(row,col),row,col);
int deleted = 0;
// loop through all of left rows
    for (int i=0; i<left.getRowCount(); i++){
        if (greaterFlag) { // add greater than or equal to mean +/- n*STD
            if (gdv.getValue(left.getValueAt(i,2))>=leftDecisionValue) { // add row if >=
                tempLeft.addRow(names);
                // copy row information from left -> tempLeft
                tempLeft.setValueAt(left.getValueAt(i,0),tempLeft.getRowCount()-1,0);
                tempLeft.setValueAt(left.getValueAt(i,1),tempLeft.getRowCount()-1,1);
                tempLeft.setValueAt(left.getValueAt(i,2),tempLeft.getRowCount()-1,2);
            }
            else { // must not meet the criteria
                // remove undesired rows
                tempDep.removeRow(i-deleted);
                deleted ++;
            }
        } else { // add less than or equal to mean +/- n*STD
            if (gdv.getValue(left.getValueAt(i,2))<=leftDecisionValue) { // add row if <=
                tempLeft.addRow(names);
                // copy row information from left -> tempLeft
                tempLeft.setValueAt(left.getValueAt(i,0),tempLeft.getRowCount()-1,0);
                tempLeft.setValueAt(left.getValueAt(i,1),tempLeft.getRowCount()-1,1);
                tempLeft.setValueAt(left.getValueAt(i,2),tempLeft.getRowCount()-1,2);
            }
            else { // must not meet the criteria
                // remove undesired rows
                tempDep.removeRow(i-deleted);
                deleted ++;
            }
        }
    }
deleted = 0;
// loop through top rows
for (int i=0; i<top.getRowCount(); i++) {
    if (greaterFlag) { // print greater than or equal to mean +/- n*STD
        if (gdv.getValue(top.getValueAt(i,2))>=topDecisionValue) { // add row if >=
            tempTop.addRow(names);
            // copy row information from top -> tempTop
            tempTop.setValueAt(top.getValueAt(i,0),tempTop.getRowCount()-1,0);
            tempTop.setValueAt(top.getValueAt(i,1),tempTop.getRowCount()-1,1);
            tempTop.setValueAt(top.getValueAt(i,2),tempTop.getRowCount()-1,2);
        }
        else {
            // Disable autoCreateColumnsFromModel to prevent
            // the reappearance of columns that have been removed but
            // whose data is still in the table model
            table.setAutoCreateColumnsFromModel(false);
            // Remove the i-index visible column and its data
            removeColumnAndData(table, i-deleted);
            deleted ++;
        }
    } else { // add less than or equal to mean +/- n*STD
        if (gdv.getValue(top.getValueAt(i,2))<=topDecisionValue) { // add row <=
            tempTop.addRow(names);
            // copy row information from top -> tempTop
            tempTop.setValueAt(top.getValueAt(i,0),tempTop.getRowCount()-1,0);
            tempTop.setValueAt(top.getValueAt(i,1),tempTop.getRowCount()-1,1);
            tempTop.setValueAt(top.getValueAt(i,2),tempTop.getRowCount()-1,2);
        }
        else {
            // Disable autoCreateColumnsFromModel to prevent

```

```

// the reappearance of columns that have been removed but
// whose data is still in the table model
table.setAutoCreateColumnsFromModel(false);

// Remove the i-index visible column and its data
removeColumnAndData(table, i-deleted);

deleted ++;
}
}

}

String windowLabel; // window label to inform user what equation was used
if (greaterFlag) // user wanted >=
    if (nTimes >= 0) // user wanted positive 'n'
        windowLabel = new String( "Dependency Threshold View (Values >= MEAN + "+nTimes+" *STD)");
    else // user wanted negative 'n'
        windowLabel = new String( "Dependency Threshold View (Values >= MEAN "+nTimes+" *STD)");
else // user wanted <=
    if (nTimes >=0) // user wanted positive 'n'
        windowLabel = new String( "Dependency Threshold View (Values <= MEAN + "+nTimes+" *STD)");
    else // user wanted negative 'n'
        windowLabel = new String( "Dependency Threshold View (Values <= MEAN "+nTimes+" *STD)");
// display the house for the user with trimmed view : null values are not need to view house.
new HouseMatrix (project,windowLabel, -1, -1, null, tempLeft, tempTop, tempDep, null, null, currentVersion);
}
}
/**
 * Removes the specified column from the table and the associated
 * call data from the table model.
 * @param JTable and int.
 */
public void removeColumnAndData(JTable table, int vColIndex) {
    MyDefaultTableModel model = (MyDefaultTableModel)table.getModel();
    TableColumn col = table.getColumnModel().getColumn(vColIndex);
    int columnModelIndex = col.getModelIndex();
    Vector data = model.getDataVector();
    Vector colIds = model.getColumnIdentifiers();

    // Remove the column from the table
    table.removeColumn(col);

    // Remove the column header from the table model
    colIds.removeElementAt(columnModelIndex);

    // Remove the column data
    for (int r=0; r<data.size(); r++) {
        Vector row = (Vector)data.get(r);
        row.removeElementAt(columnModelIndex);
    }
    model.setDataVector(data, colIds);

    // Correct the model indices in the TableColumn objects
    // by decrementing those indices that follow the deleted column
    Enumeration enum = table.getColumnModel().getColumns();
    for (; enum.hasMoreElements(); ) {
        TableColumn c = (TableColumn)enum.nextElement();
        if (c.getModelIndex() >= columnModelIndex) {
            c.setModelIndex(c.getModelIndex()-1);
        }
    }
    model.fireTableStructureChanged();
}
}
}

```

15. Cases.QFD.UserDefinedViewGUI

```

/**-----
 * @Filename: UserDefinedViewGUI.java

```

```

* @Date: 1-21-2003
* @Author: Arthur Clomera for NPS Thesis
* @Compiler: JDK 1.3.1
* Description: This is a dialog window to get information to display a
* User-defined View it collects the information needed for
* the equation  $t \geq$  or  $t \leq$  mean  $\pm$  'n' * STD (specifically 'n' > or <).
*-----
*/

package Cases.QFD;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import javax.swing.*;

/**
 * This is a dialog window to get information to display a User-defined View.
 */
public class UserDefinedViewGUI extends JDialog {
    /**
     * default constructor which call the JDialog super and initializes
     * the GUI.
     */
    public UserDefinedViewGUI() {
        super ();
        initGUI();
        this.setSize(485,300);
        this.setVisible(true);
    }
    /**
     * polymorphic constructor to specify the dialog owner.
     * @param Dialog
     */
    public UserDefinedViewGUI (Dialog owner) {
        super (owner,"Dependency Threshold View Paramenters",true);
        initGUI();
        setSize(485,300);
        setVisible(true);
    }
    /**
     * this procedure initializes the GUI and is the central location for
     * GUI components.
     */
    public void initGUI() {
        // set up components (labels, radio buttons, etc.) in GridBagLayout
        getContentPane().setLayout(new java.awt.GridBagLayout());
        getContentPane().add(okayButton, new
java.awt.GridBagConstraints(0,4,1,1,0.0,0.0,java.awt.GridBagConstraints.CENTER,java.awt.GridBagConstraints.NONE,new
java.awt.Insets(0,0,0,0),0,0));
        getContentPane().add(lesserRadioButton,
        new java.awt.GridBagConstraints(1, 3, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(0, 1, 0, 0), 0, 0));
        getContentPane().add(greaterRadioButton,
        new java.awt.GridBagConstraints(1, 2, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.NONE,
        new java.awt.Insets(0, 0, 0, 0), 0, 0));
        getContentPane().add(label2,
        new java.awt.GridBagConstraints(0, 2, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(0, 0, 4, 1), 0, 0));
        getContentPane().add(nTextField,
        new java.awt.GridBagConstraints(1, 1, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.HORIZONTAL,
        new java.awt.Insets(1, 0, 2, 43), 0, 0));
    }
}

```

```

        getContentPane().add(label1,
            new java.awt.GridBagConstraints(0, 1, 1, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.NONE,
            new java.awt.Insets(0, 0, 0, 0), 0, 0));
        getContentPane().add(label3,
            new java.awt.GridBagConstraints(0, 0, 2, 1, 0.0, 0.0, java.awt.GridBagConstraints.CENTER,
java.awt.GridBagConstraints.HORIZONTAL,
            new java.awt.Insets(0, 0, 2, 84), 0, 0));
        label1.setText("Enter 'n' (for equation above):");
        label1.setFont(new java.awt.Font("Dialog", java.awt.Font.BOLD, 12));
        nTextField.setText("");
        nTextField.setSize(new java.awt.Dimension(21, 21));
        label2.setText("View value equal to and");
        label2.setFont(new java.awt.Font("Dialog", java.awt.Font.BOLD, 12));
        // a greater than radio button
        greaterRadioButton.setText("Greater Than");
        greaterRadioButton.setActionCommand("greaterRadioButton");
        greaterRadioButton.setLabel("<html><font color=blue>Greater Than</font></html>");
        greaterRadioButton.setHorizontalTextPosition(javax.swing.SwingConstants.LEADING);
        greaterRadioButton.setFont(new java.awt.Font("SansSerif", java.awt.Font.BOLD, 11));
        greaterRadioButton.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        greaterRadioButton.setSelected(true);
        // initialize flag to greater than by default
        this.isGreaterThanOrEqualTo = true;
        // a less than radio button
        lesserRadioButton.setText("<jRadioButton2");
        lesserRadioButton.setActionCommand("lesserRadioButton");
        lesserRadioButton.setLabel("<html><font color=blue>Less Than</font></html>");
        lesserRadioButton.setHorizontalTextPosition(javax.swing.SwingConstants.LEADING);
        lesserRadioButton.setFont(new java.awt.Font("SansSerif", java.awt.Font.BOLD, 11));
        lesserRadioButton.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        // provide a radio group to control radio buttons
        radioGroup = new ButtonGroup();
        radioGroup.add(lesserRadioButton);
        radioGroup.add(greaterRadioButton);
        // a radio button handler with listeners
        RadioButtonHandler handler = new RadioButtonHandler();
        lesserRadioButton.addItemListener(handler);
        greaterRadioButton.addItemListener(handler);
        // more labels
        label3.setText("mean +/- 'n' * std ");
        label3.setFont(new java.awt.Font("Dialog", java.awt.Font.BOLD, 14));
        label3.setAlignment(java.awt.Label.RIGHT);
        // an okay button
        okayButton.setText("<jButton1");
        okayButton.setActionCommand("okayButton");
        okayButton.setLabel("OK");
        okayButton.addActionListener(new ActionListener() {public void actionPerformed(ActionEvent
e) {okayButtonActionPerformed(e);}});
    }
    /**
     * a main procedure for unit testing.
     */
    public static void main (String[] args)
    {
        UserDefinedViewGUI application = new UserDefinedViewGUI();
        application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    /**
     * this internal class handles radio button events.
     */
    private class RadioButtonHandler implements ItemListener {
        /**
         * check to see if item state changed
         */
        public void itemStateChanged(ItemEvent event) {
            // must have pressed greater than radio button
            if (event.getSource () == greaterRadioButton)

```

```

        isGreaterThan = true;
// must have pressed less than radio button
if (event.getSource() == lesserRadioButton)
    isGreaterThan = false;
    }
}

/** a label for the GUI */
private Label label1 = new Label();
/** text area for user input value */
private TextField nTextField = new TextField();
/** yet another label for the GUI */
private Label label2 = new Label();
/** a button group for greater or less than */
private ButtonGroup radioGroup;
/** greater than radio button */
private JRadioButton greaterRadioButton = new JRadioButton();
/** less than radio button */
private JRadioButton lesserRadioButton = new JRadioButton();
/** yes, yet another label for GUI */
private Label label3 = new Label();
/** boolean to match radio button events */
private boolean isGreaterThan=false;
/** an okay button */
private JButton okayButton = new JButton();

/**
 * returns the isGreaterThan state
 * @return boolean
 */
public boolean getIsGreaterThan () {
    return isGreaterThan;
}

/**
 * returns value of nValue passed by user
 * @return String
 */
public String getNValue () {
    return this.nTextField.getText();
}

/**
 * a procedure to track okay button events
 */
public void okayButtonActionPerformed(ActionEvent e) {
    double amount;
    try{
        amount = Double.valueOf(getNValue()).doubleValue();
    } catch (java.lang.NumberFormatException e2) {
        JOptionPane.showMessageDialog(this,"You must enter a numeric value","Invalid Input",JOptionPane.ERROR_MESSAGE);
        return;
    }
    this.hide();
}
}

```

I. CASES.QFD.UTIL PACKAGE

1. Cases.QFD.util.AVCTools

```

/**-----
 * Filename: AVCTool.java
 * @Date: 4-4-2003
 * @Author: Arthur Clomera for NPS Thesis
 * Compiler: JDK 1.3.1
 * Description: An AVC tool for getting secondary input and output components

```

```

*-----
**/

package Cases.QFD.util;

import java.util.Vector;
import javax.swing.*;
import Cases.GUI.avc.AVCOpenStepFrame;
public class AVCTool {
    public AVCTool(String projectName) {
        this.avcOpen = new AVCOpenStepFrame (projectName);
    }
    /** returns all secondary input components for componentID passed in */
    public Vector getSecondaryInputComponents(String componentID) {
        JComboBox tempJCB = this.avcOpen.getStepIDComboBox();
        Vector tempVector = new Vector();
        for (int i=1;i<tempJCB.getItemCount(); i++) {
            if (componentID.equalsIgnoreCase(tempJCB.getItemAt(i).toString().substring(3))) {
                tempVector.add(new
String(tempJCB.getItemAt(i).toString().substring(2,3)+tempJCB.getItemAt(i).toString().substring(4)));
            }
        }
        return tempVector;
    }
    /** returns all secondary output components for componentID passed in */
    public Vector getSecondaryOutputComponents(String componentID) {
        JComboBox tempJCB = this.avcOpen.getStepIDComboBox();
        Vector tempVector = new Vector();
        for (int i=1;i<tempJCB.getItemCount(); i++) {
            if
(componentID.equalsIgnoreCase(tempJCB.getItemAt(i).toString().substring(2,3)+tempJCB.getItemAt(i).toString().substring(4)))
                tempVector.add(new String(tempJCB.getItemAt(i).toString().substring(3)));
            }
        }
        return tempVector;
    }
}
/** a variable to hold avc open step frame */
private AVCOpenStepFrame avcOpen;
}

```

2. Cases.QFD.util.ExcelCSVLexer

```

/**-----
* Filename: ExcelCSVLexer.java
* @Date: 1-20-2003
* @Modifier: Arthur Clomera for NPS Thesis
* Compiler: JDK 1.3.1
* Description: This class is a MS Excel CSV lexer, it will read in 3 values:
* An Identifier (col 1), Short Name (col 3), and a value (col ?).
*-----
*/

/*
* Read files in comma separated value format.
* Copyright (C) 2001,2002 Stephen Ostermiller
* http://ostermiller.org/contact.pl?regarding=Java+Utilities
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*

```

```

* See COPYING.TXT for details.
*/

package Cases.QFD.util;

import java.io.*;
import java.util.Vector;
import javax.swing.*;
import Cases.CasesTitle;
import Cases.ListDialog;
import Cases.QFD.MyDefaultTableModel;
/**
 * Read files in comma separated value format as outputted by the Microsoft
 * Excel Spreadsheet program.
 * More information about this class is available from <a target="_top" href=
 * "http://ostermiller.org/Utils/ExcelCSV.html">ostermiller.org</a>.
 * <P>
 * Excel CSV is a file format used as a portable representation of a database.
 * Each line is one entry or record and the fields in a record are separated by commas.
 * <P>
 * If field includes a comma or a new line, the whole field must be surrounded with double quotes.
 * When the field is in quotes, any quote literals must be escaped by two quotes ("").
 * Text that comes after quotes that have been closed but come before the next comma will be ignored.
 * <P>
 * Empty fields are returned as as String of length zero: "". The following line has three empty
 * fields and three non-empty fields in it. There is an empty field on each end, and one in the
 * middle. One token is returned as a space.<br>
 * <pre>second,, ,fifth,</pre>
 * <P>
 * Blank lines are always ignored. Other lines will be ignored if they start with a
 * comment character as set by the setCommentStart() method.
 * <P>
 * An example of how CSVLexer might be used:
 * <pre>
 * ExcelCSVLexer shredder = new ExcelCSVLexer(System.in);
 * String t;
 * while ((t = shredder.getNextToken()) != null) {
 *     System.out.println("" + shredder.getLineNumber() + " " + t);
 * }
 * </pre>
 * <P>
 * The CSV that Excel outputs differs from the generally accepted standard CSV standard in several respects:
 * <ul><li>Leading and trailing whitespace is significant.</li>
 * <li>A backslash is not a special character and is not used to escape anything.</li>
 * <li>Quotes inside quoted strings are escaped with a double quote rather than a backslash.</li>
 * <li>Excel may convert data before putting it in CSV format:<ul>
 * <li>Tabs are converted to a single space.</li>
 * <li>New lines in the data are always represented as the unix new line. ("n")</li>
 * <li>Numbers that are greater than 12 digits may be represented in truncated
 * scientific notation form.</li></ul>
 * <li>This parser does not attempt to fix these excel conversions, but users should be aware
 * of them.</li></ul>
 */

/**
 * This class is a scanner generated by
 * <a href="http://www.jflex.de/">JFlex</a> 1.3.5
 * on 10/24/02 1:19 PM from the specification file
 * <tt>file:/home/steveo/personal/projects/java/com/Ostermiller/util/ExcelCSVLexer.lex</tt>
 * @testcase test.Cases.QFD.util.TestExcelCSVLexer
 */
public class ExcelCSVLexer extends JFrame implements Cases.CasesTitle {

    /** This character denotes the end of file */
    final public static int YYEOF = -1;

    /** initial size of the lookahead buffer */

```

```

final private static int YY_BUFFER_SIZE = 16384;

/** lexical states */
final public static int BEFORE = 1;
final public static int YY_INITIAL = 0;
final public static int COMMENT = 3;
final public static int AFTER = 2;

/**
 * Translates characters to character classes
 */
final private static String yyemap_packed =
    "\12\0\12\2\0\1\124\0\13\11\0\14\uffed3\0";

/**
 * Translates characters to character classes
 */
final private static char [] yyemap = yy_unpack_cmap(yyemap_packed);
    private RandomAccessFile output;

/**
 * Translates a state to a row index in the transition table
 */
final private static int[] yy_rowMap = {
    0, 5, 10, 15, 20, 25, 30, 35, 30, 40,
    45, 30, 50, 30, 55, 30, 60, 65, 70
};

/**
 * The packed transition table of the DFA (part 0)
 */
final private static String yy_packed0 =
    "\15\16\17\110\111\112\113\114"+
    "\115\116\117\16\17\16\17\17\120\121"+
    "\16\172\21\152\0\153\0\17"+
    "\7\03\10\122\110\112\2\0\12\3\0"+
    "\1\14\2\03\15\123\115\117\2\0\17"+
    "\1\0\121\2\02\21\3\0\110\4\0\15"+
    "\1\0";

/**
 * The transition table of the DFA
 */
final private static int[] yytrans = yy_unpack();

/* error codes */
final private static int YY_UNKNOWN_ERROR = 0;
///
// final private static int YY_ILLEGAL_STATE = 1;
final private static int YY_NO_MATCH = 2;
final private static int YY_PUSHBACK_2BIG = 3;

/* error messages for the codes above */
final private static String[] YY_ERROR_MSG = {
    "Unkown internal scanner error",
    "Internal error: unknown state",
    "Error: could not match input",
    "Error: pushback value was too large"
};

/**
 * YY_ATTRIBUTE[aState] contains the attributes of state <code>aState</code>
 */
private final static byte[] YY_ATTRIBUTE = {
    0, 0, 1, 1, 1, 1, 9, 0, 9, 1, 1, 9, 0, 9, 1, 9,
    1, 1, 1
};

```



```

/** the input device */
private Reader yy_reader;
/** column header value to track the dependency column number */
private int colHeaderValue=0;

/** the current state of the DFA */
private int yy_state;

/** the current lexical state */
private int yy_lexical_state = YYINITIAL;

/** this buffer contains the current text to be matched and is
 * the source of the yytext() string */
private char[] yy_buffer = new char[YY_BUFFERSIZE];

/** the textposition at the last accepting state */
private int yy_markedPos;

/** the textposition at the last state to be included in yytext */
private int yy_pushbackPos;

/** the current text position in the buffer */
private int yy_currentPos;

/** startRead marks the beginning of the yytext() string in the buffer */
private int yy_startRead;

/** endRead marks the last character in the buffer, that has been read
 * from input */
private int yy_endRead;

/** number of newlines encountered up to the start of the matched text */
private int yyline;

/** the number of characters up to the start of the matched text */
private int yychar;

/**
 * the number of characters from the last newline up to the start of the
 * matched text
 */
private int yycolumn;

/**
 * yy_atBOL == true <=> the scanner is currently at the beginning of a line
 */
private boolean yy_atBOL = true;

/** yy_atEOF == true <=> the scanner is at the EOF */
private boolean yy_atEOF;

/* user code: */
/**
 * Prints out tokens and line numbers from a file or System.in.
 * If no arguments are given, System.in will be used for input.
 * If more arguments are given, the first argument will be used as
 * the name of the file to use as input
 *
 * @param args program arguments, of which the first is a filename
 */
public static void main(String[] args) {
    ExcelCSVLexer shredder = new ExcelCSVLexer(0);
}

/** a method to set the colHeaderValue variable */
public void setColHeaderValue(int col) {
    this.colHeaderValue=col;
}

```

```

private String unescape(String s){
    if(s.indexOf("\", 1) == s.length()-1){
        return s.substring(1, s.length()-1);
    }
    StringBuffer sb = new StringBuffer(s.length());
    for (int i=1; i<s.length()-1; i++){
        char c = s.charAt(i);
        char c1 = s.charAt(i+1);
        if (c == "\"" && c1 == "\""){
            i++;
            sb.append("\"");
        } else {
            sb.append(c);
        }
    }
    return sb.toString();
}

private String commentDelims = "";

/**
 * Set the characters that indicate a comment at the beginning of the line.
 * For example if the string "#;!" were passed in, all of the following lines
 * would be comments:<br>
 * <pre># Comment
 * ; Another Comment
 * ! Yet another comment</pre>
 * By default there are no comments in Excel CVS files. Commas and quotes may not be
 * used to indicate comment lines.
 *
 * @param commentDelims list of characters a comment line may start with.
 */
public void setCommentStart(String commentDelims){
    this.commentDelims = commentDelims;
}

private boolean addLine = true;
private int lines = 0;
private Object[] names = {"", "", ""};
private MyDefaultTableModel CSVTable;
private MyDefaultTableModel stepCSVTable;

/**
 * Get the line number that the last token came from.
 * <p>
 * New line breaks that occur in the middle of a token are not
 * counted in the line number count.
 * <p>
 * If no tokens have been returned, the line number is undefined.
 *
 * @return line number of the last token.
 */
public int getLineNumber(){
    return lines;
}

/**
 * Creates a new scanner
 * There is also a java.io.InputStream version of this constructor.
 *
 * @param in the java.io.Reader to read input from.
 */
public ExcelCSVLexer(Reader in) {
    this.yy_reader=in;
}
/** used to import step CSV file */

```

```

public ExcelCSVLexer () {
    super ("Import CSV file into CASES");
    InputStream in;
    in = openFile();
    ExcelCSVLexer shredder = new ExcelCSVLexer(in);
    ExcelCSVLexer shredder2 = shredder;
    GetDoubleValue gdv = new GetDoubleValue();
    MatrixCalculator mc = new MatrixCalculator();
    int row=1, col=1;
    Vector tempVector = new Vector();
    Vector tempRowVector = new Vector();
    String t="";
    try {
        while (((t = shredder.getNextToken()) != null) ){
            if (row != shredder.getLineNumber()) {
                row++;
                tempVector.add(t);
                col=1;
                tempRowVector.add(tempVector);
                tempVector = new Vector();
            }
            tempVector.add(t);
            col++;
        }
    } catch (IOException e){ System.out.println(e.getMessage()); }
    tempRowVector.add(tempVector);
    stepCSVTable = new MyDefaultTableModel(row,col-1);
    mc.removeColumnIdentifiers(stepCSVTable);

    for (int i=0; i<row; i++) {
        for (int j=0; j<col-1; j++) {

            stepCSVTable.setValueAt(((Vector)tempRowVector.get(i)).get(j),i,j);

        }
    }
}

public ExcelCSVLexer(int depValue) {
    super ("Import CSV file into CASES");
    Vector colNames = new Vector();
    InputStream in;
    in = openFile();
    ExcelCSVLexer shredder = new ExcelCSVLexer(in);
    CSVTable = new MyDefaultTableModel(names,0);
    String t, t2, t3;
    Double t4;
    boolean depFlag = true;
    GetDoubleValue gdv = new GetDoubleValue();
    int lastLineNumber=0;
    try {
        while (((t = shredder.getNextToken()) != null) ){
            if (shredder.getLineNumber()==1) {
                colNames.add(t);
            } else if (depFlag ){
                if( depValue==0 ){
                    (new ListDialog(this, "Column Header List", colNames)).setVisible(true);
                }
                depFlag = false;
            }
        }
        if (lastLineNumber != shredder.getLineNumber()) { // print first line only
            if (this.colHeaderValue > 0) {
                String[] colValues=new String[50];
                for (int colLoop = 0; colLoop < this.colHeaderValue+2; colLoop++) {
                    colValues[colLoop]=shredder.getNextToken();
                }
                t2=colValues[0];
                t3=colValues[1];
                // System.out.println(this.colHeaderValue);
            }
        }
    }
}

```

```

//      System.out.println(colValues[2]+colValues[3]+colValues[4]);
//      t4=new Double(gdv.getValue(colValues[this.colHeaderValue+1]));
//      } else {
//          t2=shredder.getNextToken(); // get 2nd token
//          t3=shredder.getNextToken(); // get 3rd token
//          t4=new Double(shredder.getLineNumber()); // a generic value
//      }
//          if (shredder.getLineNumber()>1) { // ship first line
//      System.out.println(t+" "+t3+" "+t4);
//          Object[] tempObject = {t,t3,t4};
//          CSVTable.addRow(tempObject);
//      }
//      lastLineNumber ++;
//      }
//      } catch (IOException e){ System.out.println(e.getMessage()); }

}
/**
 * Creates a new scanner.
 * There is also java.io.Reader version of this constructor.
 *
 * @param in the java.io.InputStream to read input from.
 */
public ExcelCSVLexer(InputStream in) {
    this(new java.io.InputStreamReader(in));
}
private InputStream openFile()
{
    InputStream inputStream = System.in;
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setCurrentDirectory(CSVDIRECTORY);
    fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
    int result = fileChooser.showOpenDialog(this);
    // if user clicked Cancel button on dialog, return
    if (result == JFileChooser.CANCEL_OPTION)
        return null;
    // obtain selected file
    File fileName = fileChooser.getSelectedFile();

    // display error if file name invalid
    if (fileName == null || fileName.getName().equals(""))
        JOptionPane.showMessageDialog(this, "Invalid File Name", "Invalid File Name", JOptionPane.ERROR_MESSAGE);
    else {
        // open file
        try {
            output = new RandomAccessFile (fileName, "rw");
            inputStream = new FileInputStream(fileName);
        }
        catch (IOException ioException) {
            JOptionPane.showMessageDialog(this, "File does not exist", "Invalid File Name", JOptionPane.ERROR_MESSAGE);
        }
    }

    return inputStream;
}

/**
 * Unpacks the split, compressed DFA transition table.
 *
 * @return the unpacked transition table
 */
private static int [] yy_unpack() {
    int [] trans = new int[75];
    int offset = 0;
    offset = yy_unpack(yy_packed0, offset, trans);
    return trans;
}

```

```

/**
 * Unpacks the compressed DFA transition table.
 *
 * @param packed the packed transition table
 * @return the index of the last entry
 */
private static int yy_unpack(String packed, int offset, int [] trans) {
    int i = 0; /* index in packed string */
    int j = offset; /* index in unpacked array */
    int l = packed.length();
    while (i < l) {
        int count = packed.charAt(i++);
        int value = packed.charAt(i++);
        value--;
        do trans[j++] = value; while (--count > 0);
    }
    return j;
}

/**
 * Unpacks the compressed character translation table.
 *
 * @param packed the packed character translation table
 * @return the unpacked character translation table
 */
private static char [] yy_unpack_cmap(String packed) {
    char [] map = new char[0x10000];
    int i = 0; /* index in packed string */
    int j = 0; /* index in unpacked array */
    while (i < 18) {
        int count = packed.charAt(i++);
        char value = packed.charAt(i++);
        do map[j++] = value; while (--count > 0);
    }
    return map;
}

/**
 * Refills the input buffer.
 *
 * @return <code>>false</code>, iff there was new input.
 *
 * @exception IOException if any I/O-Error occurs
 */
private boolean yy_refill() throws IOException {

    /* first: make room (if you can) */
    if (yy_startRead > 0) {
        System.arraycopy(yy_buffer, yy_startRead,
            yy_buffer, 0,
            yy_endRead-yy_startRead);

        /* translate stored positions */
        yy_endRead -= yy_startRead;
        yy_currentPos -= yy_startRead;
        yy_markedPos -= yy_startRead;
        yy_pushbackPos -= yy_startRead;
        yy_startRead = 0;
    }

    /* is the buffer big enough? */
    if (yy_currentPos >= yy_buffer.length) {
        /* if not: blow it up */
        char[] newBuffer = new char[yy_currentPos*2];
        System.arraycopy(yy_buffer, 0, newBuffer, 0, yy_buffer.length);
        yy_buffer = newBuffer;
    }
}

```

```

/* finally: fill the buffer with new input */
int numRead = yy_reader.read(yy_buffer, yy_endRead,
                             yy_buffer.length-yy_endRead);

if (numRead < 0) {
    return true;
}
else {
    yy_endRead += numRead;
    return false;
}
}

/**
 * Closes the input stream.
 */
final public void yyclose() throws IOException {
    yy_atEOF = true; /* indicate end of file */
    yy_endRead = yy_startRead; /* invalidate buffer */

    if (yy_reader != null)
        yy_reader.close();
}

/**
 * Closes the current stream, and resets the
 * scanner to read from a new input stream.
 *
 * All internal variables are reset, the old input stream
 * <b>cannot</b> be reused (internal buffer is discarded and lost).
 * Lexical state is set to <tt>YY_INITIAL</tt>.
 *
 * @param reader the new input stream
 */
final public void yyreset(Reader reader) throws IOException {
    yyclose();
    yy_reader = reader;
    yy_atBOL = true;
    yy_atEOF = false;
    yy_endRead = yy_startRead = 0;
    yy_currentPos = yy_markedPos = yy_pushbackPos = 0;
    yyline = yychar = yycolumn = 0;
    yy_lexical_state = YY_INITIAL;
}

/**
 * Returns the current lexical state.
 */
final public int yystate() {
    return yy_lexical_state;
}

/**
 * Enters a new lexical state
 *
 * @param newState the new lexical state
 */
final public void yybegin(int newState) {
    yy_lexical_state = newState;
}

/**

```

```

    * Returns the text matched by the current regular expression.
    */
    final public String yytext() {
        return new String( yy_buffer, yy_startRead, yy_markedPos-yy_startRead );
    }

    /**
     * Returns the character at position <tt>pos</tt> from the
     * matched text.
     *
     * It is equivalent to yytext().charAt(pos), but faster
     *
     * @param pos the position of the character to fetch.
     *           A value from 0 to yylength()-1.
     *
     * @return the character at position pos
     */
    final public char yycharat(int pos) {
        return yy_buffer[yy_startRead+pos];
    }

    /**
     * Returns the length of the matched text region.
     */
    final public int yylength() {
        return yy_markedPos-yy_startRead;
    }

    /**
     * Reports an error that occurred while scanning.
     *
     * In a wellformed scanner (no or only correct usage of
     * yypushback(int) and a match-all fallback rule) this method
     * will only be called with things that "Can't Possibly Happen".
     * If this method is called, something is seriously wrong
     * (e.g. a JFlex bug producing a faulty scanner etc.).
     *
     * Usual syntax/scanner level error handling should be done
     * in error fallback rules.
     *
     * @param errorCode the code of the error message to display
     */
    private void yy_ScanError(int errorCode) {
        String message;
        try {
            message = YY_ERROR_MSG[errorCode];
        }
        catch (ArrayIndexOutOfBoundsException e) {
            message = YY_ERROR_MSG[YY_UNKNOWN_ERROR];
        }

        throw new Error(message);
    }

    ///
    // /**
    // * Pushes the specified amount of characters back into the input stream.
    // *
    // * They will be read again by then next call of the scanning method
    // *
    // * @param number the number of characters to be read again.
    // *           This number must not be greater than yylength()!
    // */
    // private void yypushback(int number) {
    //     if ( number > yylength() )

```

```

// yy_ScanError(YY_PUSHBACK_2BIG);
//
// yy_markedPos -= number;
// }

/**
 * Resumes scanning until the next regular expression is matched,
 * the end of input is encountered or an I/O-Error occurs.
 *
 * @return the next token
 * @exception IOException if any I/O-Error occurs
 */
public String getNextToken() throws IOException {
    int yy_input;
    int yy_action;

    // cached fields:
    int yy_currentPos_l;
    int yy_startRead_l;
    int yy_markedPos_l;
    int yy_endRead_l = yy_endRead;
    char [] yy_buffer_l = yy_buffer;
    char [] yycmap_l = yycmap;

    int [] yytrans_l = yytrans;
    int [] yy_rowMap_l = yy_rowMap;
    byte [] yy_attr_l = YY_ATTRIBUTE;

    while (true) {
        yy_markedPos_l = yy_markedPos;

        yy_action = -1;

        yy_startRead_l = yy_currentPos_l = yy_currentPos =
            yy_startRead = yy_markedPos_l;

        yy_state = yy_lexical_state;

        yy_forAction: {
            while (true) {

                if (yy_currentPos_l < yy_endRead_l)
                    yy_input = yy_buffer_l[yy_currentPos_l++];
                else if (yy_atEOF) {
                    yy_input = YYEOF;
                    break yy_forAction;
                }
                else {
                    // store back cached positions
                    yy_currentPos = yy_currentPos_l;
                    yy_markedPos = yy_markedPos_l;
                    boolean eof = yy_refill();
                    // get translated positions and possibly new buffer
                    yy_currentPos_l = yy_currentPos;
                    yy_markedPos_l = yy_markedPos;
                    yy_buffer_l = yy_buffer;
                    yy_endRead_l = yy_endRead;
                    if (eof) {
                        yy_input = YYEOF;
                        break yy_forAction;
                    }
                    else {
                        yy_input = yy_buffer_l[yy_currentPos_l++];
                    }
                }
            }
        }
        int yy_next = yytrans_l[ yy_rowMap_l[yy_state] + yycmap_l[yy_input] ];
    }
}

```



```

        if (yy_next == -1) break yy_forAction;
        yy_state = yy_next;

        int yy_attributes = yy_attr_l[yy_state];
        if ( (yy_attributes & 1) == 1 ) {
            yy_action = yy_state;
            yy_markedPos_1 = yy_currentPos_1;
            if ( (yy_attributes & 8) == 8 ) break yy_forAction;
        }
    }
}

// store back cached position
yy_markedPos = yy_markedPos_1;

switch (yy_action) {

    case 2:
    case 14:
    {

    }
    case 20: break;
    case 3:
    case 16:
    {

    }
    case 21: break;
    case 9:
    {
        yybegin(AFTER);
        return(yytext());
    }
    case 22: break;
    case 8:
    {
        if (addLine) {
            lines++;
            addLine = false;
        }
        yybegin(BEFORE);
        return("");
    }
    case 23: break;
    case 5:
    case 6:
    {
        addLine = true;
        yybegin(YYINITIAL);
    }
    case 24: break;
    case 10:
    case 11:
    {
        yybegin(YYINITIAL);
        addLine = true;
        return("");
    }
    case 25: break;
    case 4:
    {
        if (addLine) {
            lines++;
            addLine = false;
        }
        String text = yytext();
        if (commentDelims.indexOf(text.charAt(0)) == -1){
            yybegin(AFTER);

```

```

        return(text);
    } else {
        yybegin(COMMENT);
    }
}

    case 26: break;
    case 18:
    {
        yybegin(AFTER);
        return(unescape(yytext()));
    }

    case 27: break;
    case 15:
    {
        yybegin(BEFORE);
    }

    case 28: break;
    case 17:
    {
        if (addLine) {
            lines++;
            addLine = false;
        }
        yybegin(AFTER);
        return(unescape(yytext()));
    }

    case 29: break;
    case 13:
    {
        yybegin(BEFORE);
        return("");
    }

    case 30: break;
    default:
        if (yy_input == YYEOF && yy_startRead == yy_currentPos) {
            yy_atEOF = true;
            switch (yy_lexical_state) {
                case BEFORE:
                {
                    yybegin(YYINITIAL);
                    addLine = true;
                    return("");
                }

                case 20: break;
                default:
                    return null;
            }
        }
        else {
            yy_ScanError(YY_NO_MATCH);
        }
    }
}
}
}

public MyDefaultTableModel getCSVTable() { return CSVTable; }

public void setCSVTable(MyDefaultTableModel CSVTable) { this.CSVTable = CSVTable; }
public MyDefaultTableModel getStepCSVTable() { return stepCSVTable; }
public void setStepCSVTable(MyDefaultTableModel CSVTable) { this.stepCSVTable = CSVTable; }
}

```

3. Cases.QFD.util.GetDoubleValue

```

/**
 * -----

```

```

* @Filename: GetDoubleValue.java
* @Date: 1-22-2003
* @Author: Arthur Clomera for NPS Thesis
* Compiler: JDK 1.3.1
* Description: GetDoubleValue is a conversion utility class for string to double values.
* Many of these functions are used by the upstream and downstream calculations.
* Additionally, they are used to get values from cells of JTables and format them.
//-----
**/
package Cases.QFD.util;

/**
 * @testcase test.Cases.QFD.util.TestGetDoubleValue
 */
public class GetDoubleValue {
    public GetDoubleValue() {
    }

    /**
     * get the double value of an object
     */
    public double getValue(Object value) {
        String s;
        s = value.toString();
        return Double.valueOf(s).doubleValue();
    }

    /**
     * get the double value of an string
     */
    public double getValue(String value) {
        double returnValue=0.0;
        try {
            returnValue =Double.valueOf(value).doubleValue();
        } catch (NullPointerException e){
            System.out.println(e.getMessage());
        }
        return returnValue;
    }

    /**
     * squares the parameter x
     */
    public double square (double x) {
        return x*x;
    }
}

/**
 * roundUp: rounds double up and formats it for output
 * @return a double that is formatted
 * @param takes a double that needs to be rounded and formatted
 */
    public double roundUp( double d) {
        return ((double)(long)(d*100 + 0.5))/100;
    }
}

```

4. Cases.QFD.util.MatrixCalculator

```

/**-----
 * Filename MatrixCalculator.java
 * Matrix Calculator
 * @Author: Arthur Clomera for NPS Thesis
 * @Compiler: JDK 1.3.1
 * @since 3-8-2003
 * This class will remove columns ids from a table and transpose a table
 *

```

```

/**
package Cases.QFD.util;

import java.util.Vector;
import Cases.QFD.MyDefaultTableModel;

/**
 * @testcase test.Cases.QFD.util.TestMatrixCalculator
 */
public class MatrixCalculator {
    /**default constructor */
    public MatrixCalculator() {
    }

    /**
     * transposeTable : rotates a table clock-wise
     * @return a table model that is the transpose of the table given
     * @param a table model that needs to be rotated right
     */
    public MyDefaultTableModel transposeComponent (MyDefaultTableModel table) {
        int row = table.getRowCount();
        int col = table.getColumnCount();
        MyDefaultTableModel tempTable = new MyDefaultTableModel (col, row);

        for (int i = 0; i<col; i++) {
            for (int j = 0; j<row; j++)
                tempTable.setValueAt(table.getValueAt(j,i),i,j);
            }
        removeColumnIdentifiers(tempTable);
        return tempTable;
    }
    /**
     * removeColumnIdentifies: removes the column identifiers from the top of the table
     * @return a default table model without column identifiers
     * @param requires a table model to be passed in
     */
    public MyDefaultTableModel removeColumnIdentifiers (MyDefaultTableModel table) {
        MyDefaultTableModel tempTable = table;
        int col = table.getColumnCount();
        Vector names = new Vector();
        for (int i = 0; i<col; i++) {
            names.addElement("");
        }
        tempTable.setColumnIdentifiers(names);
        return tempTable;
    }
}

```

5. Cases.QFD.util.ObjectFileOperations

```

/**
 *-----
 * Filename: ObjectFileOperations.java
 * @Date: 2-12-2003
 * @Author: Arthur Clomera for NPS Thesis
 * @Compiler: JDK 1.3.1
 * Description: This is a consolidated file operation class.
 *-----
 */

package Cases.QFD.util;

import java.io.*;
import java.util.Hashtable;

```

```

import java.util.Vector;

/**
 * @testcase test.Cases.QFD.util.TestObjectFileOperations
 */
public class ObjectFileOperations {
    public void fileSaveActionPerformed(String fileFullPath, Hashtable objHashTable) {
        try{
            FileOutputStream objFileOut = new FileOutputStream( fileFullPath );
            ObjectOutputStream fileOut = new ObjectOutputStream( objFileOut );
            if( objHashTable.size() > 0 ){
                if( fileOut != null ){
                    fileOut.writeObject( objHashTable );
                }
                fileOut.flush();
                fileOut.close();
                objFileOut.close();
            }
        }
        catch(IOException e ){
            System.out.println("IOException_ObjFileCancel: "+e);
        }
    }
}
/**
 * Save an object type in an object Vector in file
 */
    public void fileSaveActionPerformed(String fileFullPath, Vector objVector) {
        try{
            FileOutputStream objFileOut = new FileOutputStream(fileFullPath);
            ObjectOutputStream objOut= new ObjectOutputStream( objFileOut );
            if( objVector.size()> 0 ){
                if(objOut != null )
                    objOut.writeObject(objVector);
                objOut.flush();
                objOut.close();
                objFileOut.close();
            }
        }
        catch( FileNotFoundException fe ){
            System.out.println("FileNotFoundException: "+fe);
        }
        catch( IOException e ){
            System.out.println("IOException: "+e);
        }
    }
}
/**
 * Read file and insert their contents
 * into a Vector
 *
 * @param projectPath : the path of current project
 * @param fileName : the name of the file to be loaded
 * @param objVector : the vector to store objects
 */
public Vector fileLoadActionPerformed(String projectPath, String fileName){
    File objFile = new File(projectPath, fileName);
    Vector objVector = new Vector();
    if( objFile.exists() ){
        try{
            FileInputStream fileInput = new FileInputStream( objFile );
            ObjectInputStream objIn = new ObjectInputStream( fileInput );
            if(objIn != null ){
                objVector = (Vector)objIn.readObject();
            }
            objIn.close();
            fileInput.close();
        }
    }
}

```

```

        catch( IOException e ){
            System.out.println("IOException_ObjVectorInitial: "+e);
        }
        catch( ClassNotFoundException ex ){
            System.out.println("ClassNotFoundException_ObjVectorInitial: "+ex);
        }
    }

    return objVector;
}

/**
 * default constructor : doesn't require any parameters
 */
public ObjectFileOperations() {
}
}

```

J. JOBSCHEDULE PACKAGE

1. JobSchedule.JAboutDialog

```

/**-----
 * Filename: JAboutDialog.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: JDK 1.3.1
 * Description: A basic implementation of the JDialog class. (needs implementation)
 * Modified by removing dependencies on Visual Cafe, added standard events and listeners.
 *-----
 */
package Cases.JobSchedule;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.*;

/**
 * A basic implementation of the JDialog class.
 */
public class JAboutDialog extends javax.swing.JDialog implements ActionListener, WindowListener
{
    public JAboutDialog(Frame parentFrame) {
        super(parentFrame);

        //{{ INIT_CONTROLS
        setModal(true);
        setTitle("JFC Application - About");
        getContentPane().setLayout(new GridBagLayout());
        setSize(248,94);
        setVisible(false);
        okButton.setText("OK");
        okButton.setOpaque(false);
        okButton.setActionCommand("OK");
        okButton.setMnemonic((int)'O');
        // getContentPane().add(okButton, new
com.symantec.itools.awt.GridBagConstraintsD(2,1,1,0,0,0,0,java.awt.GridBagConstraints.CENTER,java.awt.GridBagConstraints.N
ONE,new Insets(0,0,10,0),0,0));
        okButton.setBounds(98,59,51,25);
        aboutLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        aboutLabel.setText("A JFC Application");

```

```

        // getContentPane().add(aboutLabel, new
com.symantec.itoools.awt.GridBagConstraintsD(0,0,3,1,1.0,1.0,java.awt.GridBagConstraints.CENTER,java.awt.GridBagConstraints.B
OTH,new Insets(0,0,0,0,0,0));
        aboutLabel.setBounds(0,0,248,59);
        //}}

        //{{REGISTER_LISTENERS

        okButton.addActionListener(this);
        //}}
    }

/**
 * controls the visibility of the frame based upon b
 * @param boolean
 */
    public void setVisible(boolean b) {
        if (b)
        {
            Rectangle bounds = (getParent()).getBounds();
            Dimension size = getSize();
            setLocation(bounds.x + (bounds.width - size.width)/2,
                bounds.y + (bounds.height - size.height)/2);
        }
        super.setVisible(b);
    }

    public void addNotify() {
        // Record the size of the window prior to calling parents addNotify.
        Dimension d = getSize();

        super.addNotify();

        if (fComponentsAdjusted)
            return;
        // Adjust components according to the insets
        Insets insets = getInsets();
        setSize(insets.left + insets.right + d.width, insets.top + insets.bottom + d.height);
        Component[] components = getContentPane().getComponents();
        for (int i = 0; i < components.length; i++)
        {
            Point p = components[i].getLocation();
            p.translate(insets.left, insets.top);
            components[i].setLocation(p);
        }
        fComponentsAdjusted = true;
    }

    // Used for addNotify check.
    boolean fComponentsAdjusted = false;

    //{{DECLARE_CONTROLS
    javax.swing.JButton okButton = new javax.swing.JButton();
    javax.swing.JLabel aboutLabel = new javax.swing.JLabel();
    //}}

    public void windowClosing(WindowEvent event) {
        Object object = event.getSource();
        if (object == JAboutDialog.this)
            jAboutDialog_windowClosing(event);
    }
    public void windowClosed(WindowEvent event) { }
    public void windowOpened(WindowEvent event) { }
    public void windowIconified(WindowEvent event) { }
    public void windowDeiconified(WindowEvent event) { }
    public void windowActivated(WindowEvent event) { }
    public void windowDeactivated(WindowEvent event) { }
    void jAboutDialog_windowClosing(java.awt.event.WindowEvent event)

```

```

    {
        // to do: code goes here.
        jAboutDialog_windowClosing_Interaction1(event);
    }

void jAboutDialog_windowClosing_Interaction1(java.awt.event.WindowEvent event) {
    try {
        // JAboutDialog Hide the JAboutDialog
        this.setVisible(false);
    } catch (Exception e) {System.out.println(e);
    }
}

public void actionPerformed(ActionEvent event)    {
    Object object = event.getSource();
    if (object == okButton)
        okButton_actionPerformed(event);
}

void okButton_actionPerformed(java.awt.event.ActionEvent event)    {
    // to do: code goes here.
    okButton_actionPerformed_Interaction1(event);
}

void okButton_actionPerformed_Interaction1(java.awt.event.ActionEvent event) {
    try {
        // JAboutDialog Hide the JAboutDialog
        this.setVisible(false);
    } catch (Exception e) {System.out.println(e);
    }
}
}

```

2. JobSchedule.JDialog_jobskill

```

/**-----
 * Filename: JDialog_jobskill.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1
 * Description: A job skill JDialog class. (needs implementation)
 * Modified by removing dependencies on Visual Cafe, added standard events and listeners.
 *-----
 */
package Cases.JobSchedule;

import java.awt.*;
import java.awt.event.ActionListener;
import java.util.StringTokenizer;
import java.util.Vector;
import javax.swing.*;

public class JDialog_jobskill extends JDialog implements ActionListener
{
    Vector jobskill;

    public JDialog_jobskill(Frame parent)    {
        super(parent);

        //{{ INIT_CONTROLS
        setTitle("Required Skills");
        getContentPane().setLayout(null);
        setSize(388,250);
        setVisible(false);
        JButton_delete_deleteperson.setText("Exit");

```



```

        JButton_delete_deleteperson.setActionCommand("Delete");
        getContentPane().add(JButton_delete_deleteperson);
        JButton_delete_deleteperson.setBounds(146,204,96,24);
        JLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        JLabel2.setText("Required Skills");
        getContentPane().add(JLabel2);
        JLabel2.setFont(new Font("Dialog", Font.BOLD, 15));
        JLabel2.setBounds(110,24,168,24);
        getContentPane().add(JLabel3);
        JLabel3.setBounds(74,48,240,24);
        JScrollPane1.setOpaque(true);
        getContentPane().add(JScrollPane1);
        JScrollPane1.setBounds(62,72,264,108);
        JScrollPane1.getViewport().add(JTextArea1);
        JTextArea1.setBounds(0,0,261,105);
        //}}

        //{{REGISTER_LISTENERS

        JButton_delete_deleteperson.addActionListener(this);
        //}}
    }

/**
 * default constructor
 */
    public JDialog_jobskill()    {
        this((Frame)null);
    }

/**
 * constructor with a set job skill vector v
 * @parameter Vector
 */
    public JDialog_jobskill(Vector v){
        this((Frame)null);
        jobskill=new Vector();
        jobskill=v;
        this.JTextArea1.setText("");
        this.JTextArea1.append("Skill ID"+"t"+"Skill Name"+"t"+"Skill Level"+"n");
        this.JTextArea1.append("n");
        for(int i=0; i<jobskill.size(); i++){
            String s=(String) jobskill.elementAt(i);

            StringTokenizer st = new StringTokenizer(s,".");
            st.hasMoreTokens();
            String number=new String(st.nextToken());
            String name=st.nextToken();
            String level=st.nextToken();

            this.JTextArea1.append(number+"t"+name+"t"+level+"n");
        }
    }

/**
 * constructor to set dialog with sTitle
 * @param String
 */
    public JDialog_jobskill(String sTitle)    {
        this();
        setTitle(sTitle);
    }

/**
 * controls the visibility of the frame based upon b
 * @param boolean
 */
    public void setVisible(boolean b) {

```

```

        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

/**
 * main procedure for unit testing
 */
    static public void main(String[] args)    {
        (new JDialog_jobskill()).setVisible(true);
    }

/**
 * overrides super method addNotify()
 */
    public void addNotify()    {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets
        Insets insets = getInsets();
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    /**{DECLARE_CONTROLS
    javax.swing.JButton JButton_delete_deleteperson = new javax.swing.JButton();
    javax.swing.JLabel JLabel2 = new javax.swing.JLabel();
    javax.swing.JLabel JLabel3 = new javax.swing.JLabel();
    javax.swing.JScrollPane JScrollPanel = new javax.swing.JScrollPane();
    javax.swing.JTextArea JTextArea1 = new javax.swing.JTextArea();
    //}

/**
 * method to handle standard action events
 */
    public void actionPerformed(java.awt.event.ActionEvent event)    {
        Object object = event.getSource(); // delete person button pressed
        if (object == JButton_delete_deleteperson)
            JButtonDeleteDeleteperson_actionPerformed(event);
    }

/**
 * method to handle delete personnel button action
 */
    void JButtonDeleteDeleteperson_actionPerformed(java.awt.event.ActionEvent event)    {
        // to do: code goes here.
        this.setVisible(false);
    } //end performance
}

```

3. JobSchedule.JDialog_message

```

/**-----
 * Filename: JDialog_message.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera

```

```

* Compiler: updated to JDK 1.3.1
* Description: A job message JDialog class. (needs implementation)
* Modified by removing dependencies on Visual Cafe, added standard events and listeners.
*-----
**/
package Cases.JobSchedule;

import java.awt.*;
import java.awt.event.ActionListener;
import javax.swing.*;

public class JDialog_message extends javax.swing.JDialog implements ActionListener {
    public double w;
    public Weight weight;
    JFrame_manage p;
    int n;
    /**
     * constructor with link to parent
     * @param Frame
     */
    public JDialog_message(Frame parent)    {
        super(parent);

        //{{INIT_CONTROLS
        setTitle("Error");
        getContentPane().setLayout(null);
        setSize(311,122);
        setVisible(false);
        JButton_delete_deleteperson.setText("Exit");
        JButton_delete_deleteperson.setActionCommand("Delete");
        getContentPane().add(JButton_delete_deleteperson);
        JButton_delete_deleteperson.setBounds(111,84,88,24);
        JLabel3.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        JLabel3.setText("No job scheduled");
        getContentPane().add(JLabel3);
        JLabel3.setBounds(72,24,167,24);
        //}}

        //{{REGISTER_LISTENERS
        JButton_delete_deleteperson.addActionListener(this);
        //}}
    }

    /**
     * default constructor
     */
    public JDialog_message() {          this((Frame)null);    }

    /**
     * constructor with a dialog frame title of sTitle
     * @param String
     */
    public JDialog_message(String sTitle)    {
        this();
        setTitle(sTitle);
    }

    /**
     * controls the visibility of the frame based upon b
     * @param boolean
     */
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

    /**

```

```

* main procedure for unit testing
*/
    static public void main(String[] args)    {
        (new JDialog_message()).setVisible(true);
    }

/**
 * overrides the super addNotify method
 */
    public void addNotify()    {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets
        Insets insets = getInsets();
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    /**{DECLARE_CONTROLS
    javax.swing.JButton JButton_delete_deleteperson = new javax.swing.JButton();
    javax.swing.JLabel JLabel3 = new javax.swing.JLabel();
    /**}

/**
 * method to handle standard action events
 */
    public void actionPerformed(java.awt.event.ActionEvent event)    {
        Object object = event.getSource();
        if (object == JButton_delete_deleteperson) // delete personnel
            JButtonDeleteDeleteperson_actionPerformed(event);
    }

/**
 * method to execute after delete personnel event
 */
    void JButtonDeleteDeleteperson_actionPerformed(java.awt.event.ActionEvent event)    {
        // to do: code goes here.
        this.setVisible(false);
        this.dispose();
    }
}

```

4. JobSchedule.JDialog_message1

```

/**-----
 * Filename: JDialog_message1.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1
 * Description: Another job message JDialog class (needs implementation).
 * Modified by removing dependencies on Visual Cafe, added standard events and listeners.
 *-----
 */
package Cases.JobSchedule;

import java.awt.*;

```

```

import java.awt.event.ActionListener;
import javax.swing.*;

public class JDialog_message1 extends JDialog implements ActionListener {

    public double w;
    public Weight weight;
    JFrame_manage p;
    int n;

    public JDialog_message1(Frame parent)
    {
        super(parent);

        //{{INIT_CONTROLS
        setTitle("Error");
        getContentPane().setLayout(null);
        setSize(311,122);
        setVisible(false);
        JButton_delete_deleteperson.setText("Exit");
        JButton_delete_deleteperson.setActionCommand("Delete");
        getContentPane().add(JButton_delete_deleteperson);
        JButton_delete_deleteperson.setBounds(111,84,88,24);
        JLabel3.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        JLabel3.setText("No job or stakeholder assigned");
        getContentPane().add(JLabel3);
        JLabel3.setBounds(36,24,239,36);
        //}}

        //{{REGISTER_LISTENERS

        JButton_delete_deleteperson.addActionListener(this);
        //}}

    }

    /**
     * default constructor
     */
    public JDialog_message1()    {                this((Frame)null);    }

    /**
     * constructor with dialog frame title of sTitle
     * @param String
     */
    public JDialog_message1(String sTitle)    {
        this();
        setTitle(sTitle);
    }

    /**
     * controls the visibility of the frame based upon b
     * @param boolean
     */
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

    /**
     * main procedure for unit testing
     */
    static public void main(String[] args)    {
        (new JDialog_message1()).setVisible(true);
    }

    /**
     * overrides the super addNotify() method
     */

```

```

    public void addNotify()
    {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets
        Insets insets = getInsets();
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    //{DECLARE_CONTROLS
    javax.swing.JButton JButton_delete_deleteperson = new javax.swing.JButton();
    javax.swing.JLabel JLabel3 = new javax.swing.JLabel();
    //}}

/**
 * a method to handle standard action events
 */
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == JButton_delete_deleteperson) // delete personnel
            JButtonDeleteDeleteperson_actionPerformed(event);
    }

/**
 * method to execute upon delete personnel action event
 */
    void JButtonDeleteDeleteperson_actionPerformed(java.awt.event.ActionEvent event)
    {
        // to do: code goes here.
        this.setVisible(false);

        this.dispose();
    }
}

```

5. JobSchedule.JDialog_weight

```

/**-----
 * Filename: JDialog_weight.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1 and removed Visual Cafe' dependencies
 * Description: A job weight class. (needs implementation)
 * Modified by removing dependencies on Visual Cafe, added standard events and listeners.
 *-----
 */
package Cases.JobSchedule;

import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.*;

public class JDialog_weight extends JDialog implements WindowListener
{
    public JDialog_weight(Frame parentFrame) {
        super(parentFrame);
    }
}

```

```

        //{{INIT_CONTROLS
        setModal(true);
        setTitle("JFC Application - About");
        getContentPane().setLayout(new GridBagLayout());
        setSize(416,162);
        setVisible(false);
        //}}

        //{{REGISTER_LISTENERS
        this.addWindowListener(this);
        //}}
    }

    /**
     * controls the visibility of the frame based upon b
     * @param boolean
     */
    public void setVisible(boolean b) {
        if (b)
        {
            Rectangle bounds = (getParent()).getBounds();
            Dimension size = getSize();
            setLocation(bounds.x + (bounds.width - size.width)/2,
                        bounds.y + (bounds.height - size.height)/2);
        }

        super.setVisible(b);
    }

    public void addNotify()
    {
        // Record the size of the window prior to calling parents addNotify.
        Dimension d = getSize();

        super.addNotify();

        if (fComponentsAdjusted)
            return;
        // Adjust components according to the insets
        Insets insets = getInsets();
        setSize(insets.left + insets.right + d.width, insets.top + insets.bottom + d.height);
        Component[] components = getContentPane().getComponents();
        for (int i = 0; i < components.length; i++)
        {
            Point p = components[i].getLocation();
            p.translate(insets.left, insets.top);
            components[i].setLocation(p);
        }
        fComponentsAdjusted = true;
    }

    // Used for addNotify check.
    boolean fComponentsAdjusted = false;

    //{{DECLARE_CONTROLS
    //}}

    public void windowClosing(java.awt.event.WindowEvent event) {
        Object object = event.getSource();
        if (object == JDialog_weight.this)
            jAboutDialog_windowClosing(event);
    }
    public void windowClosed(WindowEvent event) { }
    public void windowOpened(WindowEvent event) { }
    public void windowIconified(WindowEvent event) { }
    public void windowDeiconified(WindowEvent event) { }
    public void windowActivated(WindowEvent event) { }
    public void windowDeactivated(WindowEvent event) { }
    void jAboutDialog_windowClosing(java.awt.event.WindowEvent event) {

```

```

        // to do: code goes here.
        jAboutDialog_windowClosing_Interaction1(event);
    }

    void jAboutDialog_windowClosing_Interaction1(java.awt.event.WindowEvent event) {
        try {
            // JAboutDialog Hide the JAboutDialog
            this.setVisible(false);
        } catch (Exception e) {System.out.println(e);
        }
    }
}
}

```

6. JobSchedule.JDialog_weit

```

/**-----
 * Filename: JDialog_weit.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1 and removed Visual Cafe' dependencies
 * Description: A job weit class. (needs implementation)
 * Modified by removing dependencies on Visual Cafe, added standard events and listeners.
 *-----
 */
package Cases.JobSchedule;

import java.awt.*;
import java.awt.event.ActionListener;
import javax.swing.*;

public class JDialog_weit extends JDialog implements ActionListener{
    public double w;
    public Weight weight;
    JFrame_manage p;
    int n;

    public JDialog_weit(Frame parent)    {
        super(parent);

        //{{INIT_CONTROLS
        setTitle("Weight");
        getContentPane().setLayout(null);
        setSize(388,171);
        setVisible(false);
        JButton_delete_deleteperson.setText("Done");
        JButton_delete_deleteperson.setActionCommand("Delete");
        getContentPane().add(JButton_delete_deleteperson);
        JButton_delete_deleteperson.setBounds(156,132,76,24);
        JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        JLabel1.setText("Weight: ");
        getContentPane().add(JLabel1);
        JLabel1.setFont(new Font("Dialog", Font.BOLD, 16));
        JLabel1.setBounds(36,24,132,24);
        JLabel2.setText("(from 0.0 to 1.0)");
        getContentPane().add(JLabel2);
        JLabel2.setBounds(144,24,168,24);
        getContentPane().add(JTextField1);
        JTextField1.setBounds(96,84,192,24);
        getContentPane().add(JLabel3);
        JLabel3.setBounds(96,48,192,24);
        //}}

        //{{REGISTER_LISTENERS

```



```

        JButton_delete_deleteperson.addActionListener(this);
        //}}
    }

/**
 * default constructor
 */
    public JDialog_weit() {
        this((Frame)null);
    }

/**
 * constructor with p1 frame manager, weight1 weight, and n1
 * @param JFrame_manage, Weight, and int
 */
    public JDialog_weit(JFrame_manage p1, Weight weight1, int n1) {
        this((Frame)null);
        weight=weight1;
        p=p1;
        n=n1;
    }

/**
 * constructor with dialog title of sTitle
 * @param String
 */
    public JDialog_weit(String sTitle) {
        this();
        setTitle(sTitle);
    }

/**
 * controls the visibility of the frame based upon b
 * @param boolean
 */
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

    static public void main(String[] args) {
        (new JDialog_weit()).setVisible(true);
    }

    public void addNotify() {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets
        Insets insets = getInsets();
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    //{{DECLARE_CONTROLS
    javax.swing.JButton_delete_deleteperson = new javax.swing.JButton();
    javax.swing.JLabel JLabel1 = new javax.swing.JLabel();
    javax.swing.JLabel JLabel2 = new javax.swing.JLabel();
    javax.swing.JTextField JtextField1 = new javax.swing.JTextField();
    javax.swing.JLabel JLabel3 = new javax.swing.JLabel();
    //}}

```

```

    public void actionPerformed(java.awt.event.ActionEvent event)    {
        Object object = event.getSource();
        if (object == JButton_delete_deleteperson)
            JButtonDeleteDeleteperson_actionPerformed(event);
    }

    void JButtonDeleteDeleteperson_actionPerformed(java.awt.event.ActionEvent event)    {
        // to do: code goes here.
        Double d=new Double(((String)this.JTextField1.getText()).trim());
        w=d.doubleValue();
        //int i=Integer.parseInt(((String)this.JTextField1.getText()).trim());
        if(0.0<=w&&w<=1.0){
            weight.put_w(w);
            if(n==1){
                p.sortbyD_E();
                p.datavalu();
            }
            else{
                p.sortbyD_S();
                p.datavalu();
            }
            this.setVisible(false);
        }
        dispose();
        this.dispose();
    }
    else{
        this.JLabel3.setText("Invalid Value");
    }
}
}

```

7. JobSchedule.JDialog_assignjob

```

/**-----
 * Filename: JDialog_assignjob.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1
 * Description: An assign job JFrame class. (needs implementation improvements)
 * Modified by removing dependencies on Visual Cafe, added standard events and listeners.
 *-----
 */
package Cases.JobSchedule;

import java.awt.*;
import java.awt.event.ActionListener;
import java.util.Calendar;
import java.util.StringTokenizer;
import java.util.Vector;
import javax.swing.*;
import Cases.CasesFrame;
import Cases.Personnel;
import Cases.StepContent;
/**
 *      A basic implementation of the JFrame class.
 */
public class JFrame_assignjob extends JFrame implements ActionListener {
    Cursor waitCursor = new Cursor(Cursor.WAIT_CURSOR);
    Cursor defaultCursor = new Cursor(Cursor.DEFAULT_CURSOR);

    public Vector majorMinorJobs = new Vector();
    Vector person_queue = new Vector();

```

```

Vector job_queue,work_queue;
    Vector    personskill_queue,jobskill_queue;
    Vector result;
    int index, grade, step, n, year, month, date;
    String jobname;

    StepContent job;
    Vector p_q, pk_q, w_q;
    Vector job_pool = new Vector();
    CasesFrame parent;

    /**
    * default constructor
    */
    public JFrame_assignjob() {
        result=new Vector();
        step=1;
        n=0;

        //{ INIT_CONTROLS
        setTitle("Job Assignment");
        getContentPane().setLayout(null);
        setSize(492,532);
        setVisible(false);
        JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        JLabel1.setText("Job Assignment");
        getContentPane().add(JLabel1);
        JLabel1.setFont(new Font("Dialog", Font.BOLD, 16));
        JLabel1.setBounds(150,12,192,24);
        JButton1.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
        JButton1.setText("1. Filter by Security Level");
        JButton1.setActionCommand("by priority");
        getContentPane().add(JButton1);
        JButton1.setBounds(168,264,276,24);
        JButton2.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
        JButton2.setText("2. Filter by Required Skills");
        JButton2.setActionCommand("by skills");
        getContentPane().add(JButton2);
        JButton2.setBounds(168,300,276,24);
        JButton3.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
        JButton3.setText("3. Assign this Job");
        JButton3.setActionCommand("assign the job");
        getContentPane().add(JButton3);
        JButton3.setBounds(168,336,276,24);
        JButton4.setText("Exit");
        JButton4.setActionCommand("Save and Exit");
        getContentPane().add(JButton4);
        JButton4.setBounds(72,492,372,24);
        JScrollPane1.setOpaque(true);
        getContentPane().add(JScrollPane1);
        JScrollPane1.setBounds(72,372,372,108);
        JScrollPane1.getViewport().add(JTextArea2);
        JTextArea2.setBounds(0,0,369,105);
        JLabel2.setText("Security Level");
        getContentPane().add(JLabel2);
        JLabel2.setBounds(24,108,132,24);
        JLabel7.setText("Job ID");
        getContentPane().add(JLabel7);
        JLabel7.setBounds(24,72,84,28);
        JLabel8.setText("Deadline");
        getContentPane().add(JLabel8);
        JLabel8.setBounds(24,144,108,28);
        JLabel9.setText("Estimated Duration");
        getContentPane().add(JLabel9);
        JLabel9.setBounds(24,180,120,24);
        getContentPane().add(JTextField1);
        JTextField1.setBounds(168,180,276,28);
        getContentPane().add(JTextField2);

```

```

        JTextField2.setBounds(168,144,276,28);
        getContentPane().add(JTextField3);
        JTextField3.setBounds(168,108,276,28);
        getContentPane().add(JTextField4);
        JTextField4.setBounds(168,72,276,28);
        JLabel3.setText("Required Skills");
        getContentPane().add(JLabel3);
        JLabel3.setBounds(24,216,120,24);
        JButton5.setText("Required Skills");
        JButton5.setActionCommand("Job skill");
        getContentPane().add(JButton5);
        JButton5.setBounds(168,216,276,24);
    //}}

    //{{INIT_MENU
    //}}

    //{{REGISTER_LISTENERS
    JButton4.addActionListener(this);
    JButton1.addActionListener(this);
    JButton2.addActionListener(this);
    JButton3.addActionListener(this);
    JButton5.addActionListener(this);
    //}}

    //{{REGISTER_LISTENERS
    //SymAction lSymAction = new SymAction();
    //JButton3.addActionListener(lSymAction);
    //}}

    }

    /**
    * constructor with link to parent1, using the person_queue1, job_queue1, and job_pool1
    * @param CasesFrame, Vector, Vector, Vector
    */
    public JFrame_assignjob(CasesFrame parent1,Vector person_queue1,Vector job_queue1, Vector job_pool1)    {
        this();
        parent=parent1;
        this.person_queue=person_queue1; job_queue=job_queue1;
        job_pool=job_pool1;
        this.job=(StepContent) job_queue.firstElement();
        String name=job.getStepName();
        this.jobname=name;

        String sk1=new String();
        this.JTextField4.setText(job.getStepName());
        this.JTextField3.setText(sk1.valueOf(job.getSecurityLevel()));
        this.JTextField2.setText(job.getDeadline());
        this.JTextField1.setText(sk1.valueOf(job.getDuration()));
        //cleanperson_job();
    }

    /**
    * constructor with JFrame with the title of sTitle
    * @param String
    */
    public JFrame_assignjob(String sTitle)    {
        this();
        setTitle(sTitle);
    }

    /**
    * controls the visibility of the frame based upon b
    * @param boolean
    */
    public void setVisible(boolean b) {
        if (b)

```

```

        setLocation(50, 50);
        super.setVisible(b);
    }

    /**
     * main procedure for unit testing
     */
    static public void main(String[] args) {
        (new JFrame_assignjob()).setVisible(true);
    }

    /**
     * overrides the super addNotify() method
     */
    public void addNotify() {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();
        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    /**{DECLARE_CONTROLS
    javax.swing.JLabel JLabel1 = new javax.swing.JLabel();
    javax.swing.JButton JButton1 = new javax.swing.JButton();
    javax.swing.JButton JButton2 = new javax.swing.JButton();
    javax.swing.JButton JButton3 = new javax.swing.JButton();
    javax.swing.JButton JButton4 = new javax.swing.JButton();
    javax.swing.JScrollPane JScrollPane1 = new javax.swing.JScrollPane();
    javax.swing.JTextArea JTextArea2 = new javax.swing.JTextArea();
    javax.swing.JLabel JLabel2 = new javax.swing.JLabel();
    javax.swing.JLabel JLabel7 = new javax.swing.JLabel();
    javax.swing.JLabel JLabel8 = new javax.swing.JLabel();
    javax.swing.JLabel JLabel9 = new javax.swing.JLabel();
    javax.swing.JTextField JTextField1 = new javax.swing.JTextField();
    javax.swing.JTextField JTextField2 = new javax.swing.JTextField();
    javax.swing.JTextField JTextField3 = new javax.swing.JTextField();
    javax.swing.JTextField JTextField4 = new javax.swing.JTextField();
    javax.swing.JLabel JLabel3 = new javax.swing.JLabel();
    javax.swing.JButton JButton5 = new javax.swing.JButton();
    //}

    /**{DECLARE_MENUS
    //}

    /**
     * a method to handle standard action events
     */
    public void actionPerformed(java.awt.event.ActionEvent event) {
        Object object = event.getSource();
        if (object == JButton4) // button4 pressed
            JButton4_actionPerformed(event);
        else if (object == JButton1) // button1 pressed

```

```

        JButton1_actionPerformed(event);
    else if (object == JButton2) // button2 pressed
        JButton2_actionPerformed(event);
    else if (object == JButton3) // button3 pressed
        JButton3_actionPerformed(event);
    else if (object == JButton5) // button5 pressed
        JButton5_actionPerformed(event);
    }

/**
 * button4 action
 */
    void JButton4_actionPerformed(java.awt.event.ActionEvent event) {
        setVisible(false);
    dispose();
    }

    void Filterbysecurity(){
        p_q=new Vector();
        int security=job.getSecurityLevel();
        for(int i=0; i<this.person_queue.size(); i++){
            Personnel person=(Personnel) this.person_queue.elementAt(i);
            if(security<=person.getSecurityLevel()){
                p_q.addElement(person);
            }
        }
    } //end for
} //end Filter

int getmonthbyint(String month){
    if(month.equals("January")){
        return 1;
    }
    else{
        if(month.equals("February")){
            return 2;
        }
        else{
            if(month.equals("March")){
                return 3;
            }
            else{
                if(month.equals("April")){
                    return 4;
                }
                else{
                    if(month.equals("May")){
                        return 5;
                    }
                    else{
                        if(month.equals("June")){
                            return 6;
                        }
                        else{
                            if(month.equals("July")){
                                return 7;
                            }
                            else{
                                if(month.equals("August")){
                                    return 8;
                                }
                                else{
                                    if(month.equals("September")){
                                        return 9;
                                    }
                                    else{
                                        if(month.equals("October")){
                                            return 10;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```



```

        String name2=((String)st2.nextToken()).trim();
        int level2=Integer.parseInt(((String)st2.nextToken()).trim());
        if(number1==number2&&level1>=level2){
            grade++;
        }
    }//end for(k)

    }//end for(j)
    int id=Integer.parseInt(p.getID());
    Result r=new Result(id,grade);
    this.result.addElement(r);

    }//end for(i)
    sortresult();
}
catch( Exception e){}
}

void sortresult() {

    Result min, tmp;
    for(int i=0; i<this.result.size(); i++){
        min=(Result) this.result.elementAt(i);
        for(int j=i+1; j<this.result.size(); j++){
            Result r1=(Result) this.result.elementAt(j);

            if(min.get_grade()<r1.get_grade()){
                tmp=min;
                min=r1;
                this.result.setElementAt(tmp, j);
            }//end if()
        }//end for(j)

        this.result.setElementAt(min, i);
    }//end for(i)
} //end sort

Personnel seachbyid(int id) {
    try{
        for(int j=0; j<p_q.size(); j++){
            Personnel p=(Personnel) p_q.elementAt(j);
            int id1=Integer.parseInt(p.getID());
            if(id==id1){
                return p;
            }
        }
    }
}
catch( Exception e){}
return null;
} //end seach

void display1() {

    this.JTextArea2.setText("");
    this.JTextArea2.append("Stakeholder ID"+"t"+"Security Level"+"n");
    String sk=new String();
    for(int i=0; i<p_q.size(); i++){
        Personnel p=(Personnel) p_q.elementAt(i);
        this.JTextArea2.append(sk.valueOf(p.getID()+"t");
        this.JTextArea2.append(sk.valueOf(p.getSecurityLevel()+"n");
    }
}

void display2() {

    this.JTextArea2.setText("");
    this.JTextArea2.append("Stakeholder ID"+"t"+"Match Number of Required Skills"+"n");
    String s=new String();

```

```

        for(int i=0; i<this.result.size(); i++){
            Result r=(Result) this.result.elementAt(i);
            this.JTextArea2.append(s.valueOf(r.get_id()+"t"));
            this.JTextArea2.append(s.valueOf(r.get_grade()+"\n"));
        }
    }

    void display3(int pid) {
        Personnel p=searchbyid(pid);
        Vector v=p.getMajorJobs();

        StepContent step1=(StepContent) v.elementAt(0);
        StepContent step2=null;
        if(v.size()==2){
            step2=(StepContent) v.elementAt(1);
        }

        this.JTextArea2.setText("");
        this.JTextArea2.append("Stakeholder ID"+"t"+"Job ID"+"t"+"Earliest Start Time"
            +"t"+"Estimated Duration"+"n");
        this.JTextArea2.append("");
        this.JTextArea2.append(p.getID()+"t"+ step1.getStepName()+"t"+step1.getStartTime()+"t"+step1.getDuration()
            +"n");
        if(step2!=null){
            this.JTextArea2.append(p.getID()+"t"+step2.getStepName()+"t"+step2.getStartTime()+"t"+step2.getDuration()+"n");
        }
    }

    void JButton1_actionPerformed(java.awt.event.ActionEvent event) {
        this.setCursor(waitCursor);
        if(step==1){
            step++;
            Filterbysecurity();
            display1();
        }
        this.setCursor(defaultCursor);
    }

    void JButton2_actionPerformed(java.awt.event.ActionEvent event) {
        this.setCursor(waitCursor);
        if(step==2){
            step++;
            Filterbyskills();
            display2();
        }
        Calendar c= Calendar.getInstance();

        String s = c.getTime().toString();
        getdate(s);
        this.setCursor(defaultCursor);
    }

    void JButton3_actionPerformed(java.awt.event.ActionEvent event) {
        this.setCursor(waitCursor);
        if(step==3||step==4){
            step++;
            int num=0;
            if(n==0){
                num=assing_Performed();
                if(num==1){
                    parent.savePersonnelVector(this.person_queue);
                    String jobName = (String)((StepContent)this.job).getStepName();
                    parent.saveStepContentVector(this.job_pool,this.majorMinorJobs.jobName);
                    // parent.saveCompContents(this.majorMinorJobs, jobName);
                }
            }
        }
    }

```

```

        n++;
    }
    }
    this.setCursor(defaultCursor);
} //end

int assing_Performed() {
    Calendar c= Calendar.getInstance();

    int duration, personid;
    String jobname;

    duration=this.job.getDuration();
    jobname=this.job.getStepName();
    for(int i=0; i<this.result.size(); i++){
        personid=((Result) this.result.elementAt(i)).get_id();
        Personnel p=searchbyid(personid);
        Vector v1=p.getMajorJobs();
        Vector v2=p.getMinorJobs();
        String s=c.getTime().toString();
        if(v1.size()<2){
            //set the job's status
            job.setStatus("Assigned");
            //set the job's starttime
            job.setStartTime(s);
            //save job
            setjob_pool(jobname, this.job);
            //set the person's marjor job
            ((StepContent)this.job).setRealStartTime(c.getTime().toString());
            v1.addElement(this.job);

            p.setMajorJobs(v1);
            Vector newVec = (Vector) p.getMajorJobs();
            this.majorMinorJobs.addElement(p);
            //save the person
            setperson_queue(personid, p);

            //remove the job
            job_queue.removeElement(this.job);

            display3(personid);
            return 1;
        } //end if
        else{
            //set the person's minor job
            ((StepContent)this.job).setRealStartTime(c.getTime().toString());
            v2.addElement(this.job);
            //set Minor job
            p.setMinorJobs(v2);
            this.majorMinorJobs.addElement(p);
            //save the person
            setperson_queue(personid, p);
            return 1;
        }
    } //end for
    return 0;
} //end assing

void setperson_queue(int personid, Personnel p1) {
    try{
        for(int i=0; i<this.person_queue.size(); i++){
            Personnel p=(Personnel) this.person_queue.elementAt(i);
            if(personid==Integer.parseInt(p.getID())){
                this.person_queue.setElementAt(p1, i);
            }
        } //end for_loop
    }
}
catch(Exception e){

```

```

        e.printStackTrace();
    }
}

void setjob_pool(String jobname, StepContent step1) {
    for(int i=0; i<job_pool.size(); i++) {
        StepContent step=(StepContent) job_pool.elementAt(i);
        if(jobname.equals(step.getStepName())){
            job_pool.setElementAt(step1, i);
        }
    }
}

void cleanperson_job() {
    try{
        for(int i=0; i<this.person_queue.size(); i++){
            Personnel p=(Personnel) this.person_queue.elementAt(i);
            Vector v1=p.getMajorJobs();
            Vector v2=p.getMinorJobs();

            v1.removeAllElements();
            p.setMajorJobs(v1);
            v2.removeAllElements();
            p.setMinorJobs(v2);
            int personid=Integer.parseInt(p.getID());
            setperson_queue(personid, p);
        }
        parent.savePersonnelVector(this.person_queue);
    }
    catch( Exception e){}
}

void JButton5_actionPerformed(java.awt.event.ActionEvent event) {
    Vector v=(Vector) job.getSkill();
    (new JDialog_jobskill(v)).setVisible(true);
}
}

```

8. JobSchedule.JDialog_manager

```

/**-----
 * Filename: JDialog_manage.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * @Modified by: Arthur B. Clomera
 * Compiler: updated to JDK 1.3.1
 * Description: A job management JFrame class. (needs implementation)
 * Modified by removing dependencies on Visual Cafe, added standard events and listeners.
 *-----
 */
package Cases.JobSchedule;

import java.awt.*;

import java.awt.event.ActionListener;

import java.awt.event.FocusEvent;

import java.awt.event.FocusListener;

import java.awt.event.ItemListener;

```

```

import java.util.StringTokenizer;

import java.util.Vector;

import javax.swing.*;

import javax.swing.table.DefaultTableModel;

import Cases.StepContent;

/*
    A basic implementation of the JFrame class.
*/
//import com.symantec.itools.javax.swing.JButtonGroupPanel;

public class JFrame_manage extends JFrame implements ActionListener, ItemListener, FocusListener
{
    Vector job_pool, job_queue1, job_queue2, job_queue;
    double w;
    Weight weight;
    //TableModel mytable;
    int year=0, month=0, date=0;

    /**
     * default constructor
     */
    public JFrame_manage()
    {
        weight=new Weight();
        job_queue1=new Vector();
        job_queue2=new Vector();

        //initai weight value
        w=0.5;

        //{{INIT_CONTROLS
        setTitle("Job Scheduling");
        getContentPane().setLayout(null);
        setSize(596,415);
        setVisible(false);
        JLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        JLabel1.setText("Job Scheduling Policy");
        getContentPane().add(JLabel1);
        JLabel1.setFont(new Font("Dialog", Font.BOLD, 16));
        JLabel1.setBounds(190,24,216,36);
        JButton5.setText("Exit");
        JButton5.setActionCommand("Save and Exit");
        getContentPane().add(JButton5);
        JButton5.setBounds(256,372,92,24);
        getContentPane().add(JLabel6);
        JLabel6.setBounds(24,72,108,24);
        JScrollPane2.setOpaque(true);
        getContentPane().add(JScrollPane2);
        JScrollPane2.setBounds(60,132,480,216);
        JScrollPane2.getViewport().add(JTable1);
        JTable1.setBounds(0,0,477,213);
        getContentPane().add(JComboBox1);
        JComboBox1.setBounds(144,72,396,36);
        //}}

        //{{INIT_MENUS
        //}}

        //{{REGISTER_LISTENERS

        JButton5.addActionListener(this);
        JComboBox1.addItemListener(this);

```

```

        JScrollPane2.addFocusListener(this);
        JTable1.addFocusListener(this);
        //}}
        JComboBox1.addItem("Select a Policy");
        JComboBox1.addItem("High Priority First");
        JComboBox1.addItem("Minimum Deadline First (Min_D)");
        JComboBox1.addItem("Minimum Estimated Duration First (Min_E)");
        JComboBox1.addItem("Minimum Earliest Start Time First (Min_S)");
        JComboBox1.addItem("Minimum Laxity First (Min_L)");
        JComboBox1.addItem("Min_D*w + Min_E*(1-w)");
        JComboBox1.addItem("Min_D*w + Min_S*(1-w)");

    }

    /**
     * constructor with job_queue1
     * @param Vector
     */
    public JFrame_manage(Vector job_queue1) {
        this();
        job_queue=job_queue1;
        datavalu();
        //display();
    }

    /**
     * constructor with a frame title sTitle
     * @param String
     */
    public JFrame_manage(String sTitle) {
        this();
        setTitle(sTitle);
    }

    /**
     * method to control the visibility based upon b
     * @param boolean
     */
    public void setVisible(boolean b) {
        if (b)
            setLocation(50, 50);
        super.setVisible(b);
    }

    /**
     * main procedure for unit testing
     */
    static public void main(String[] args) {
        (new JFrame_manage()).setVisible(true);
    }

    /**
     * override the super addNotify() method
     */
    public void addNotify() {
        // Record the size of the window prior to calling parents addNotify.
        Dimension size = getSize();

        super.addNotify();

        if (frameSizeAdjusted)
            return;
        frameSizeAdjusted = true;

        // Adjust size of frame according to the insets and menu bar
        Insets insets = getInsets();
        javax.swing.JMenuBar menuBar = getRootPane().getJMenuBar();

```

```

        int menuBarHeight = 0;
        if (menuBar != null)
            menuBarHeight = menuBar.getPreferredSize().height;
        setSize(insets.left + insets.right + size.width, insets.top + insets.bottom + size.height + menuBarHeight);
    }

    // Used by addNotify
    boolean frameSizeAdjusted = false;

    //{ {DECLARE_CONTROLS
    javax.swing.JLabel JLabel1 = new javax.swing.JLabel();
    javax.swing.JButton JButton5 = new javax.swing.JButton();
    javax.swing.JLabel JLabel6 = new javax.swing.JLabel();
    javax.swing.JScrollPane JScrollPane2 = new javax.swing.JScrollPane();
    javax.swing.JTable JTable1 = new javax.swing.JTable();
    javax.swing.JComboBox JComboBox1 = new javax.swing.JComboBox();
    //} }

    //{ {DECLARE_MENUS
    //} }

    //creat Jtable
    void datavalu(){

        DefaultTableModel mymodel;
        int size=job_queue.size();
        String[][] datavalue=new String[size][6];
        String[] header=new String[6];

        header[0]="Job ID";
        header[1]="Priority";
        header[2]="Deadline";
        header[3]="Estimated Duration";
        header[4]="Earliest Starttime";
        header[5]="Laxity";

        for(int i=0; i<size; i++){
            String s=new String();
            StepContent step=(StepContent)job_queue.elementAt(i);
            datavalue[i][0]=step.getStepName();
            datavalue[i][1]=s.valueOf(step.getPriority());
            datavalue[i][2]=step.getDeadline();
            datavalue[i][3]=s.valueOf(step.getDuration());
            datavalue[i][4]=step.getStartTime();

            //caculation laxity
            int laxity, y1,y2,m1,m2,d1,d2;

            String deadline=step.getDeadline();
            getdate(deadline);
            y1=0; m1=0; d1=0;
            y1=this.year; m1=this.month; d1=this.date;

            String starttime=step.getStartTime();
            getdate(starttime);
            y2=0; m2=0; d2=0;
            y2=this.year; m2=this.month; d2=this.date;
            laxity=((y1-y2)*360+(m1-m2)*30+(d1-d2))-step.getDuration();

            datavalue[i][5]=s.valueOf(laxity);

        }
        mymodel=new DefaultTableModel(datavalue, header);
        JTable1.setModel(mymodel);
    }
}

```

```

/**
 * function to return month from string to int
 * @param String
 * @return int
 */
int getmonthbystring(String month){
    if(month.equals("January")){
        return 1;
    }
    else{
        if(month.equals("February")){
            return 2;
        }
        else{
            if(month.equals("March")){
                return 3;
            }
            else{
                if(month.equals("April")){
                    return 4;
                }
                else{
                    if(month.equals("May")){
                        return 5;
                    }
                    else{
                        if(month.equals("June")){
                            return 6;
                        }
                        else{
                            if(month.equals("July")){
                                return 7;
                            }
                            else{
                                if(month.equals("August")){
                                    return 8;
                                }
                                else{
                                    if(month.equals("September")){
                                        return 9;
                                    }
                                    else{
                                        if(month.equals("October")){
                                            return 10;
                                        }
                                        else{
                                            if(month.equals("November")){
                                                return 11;
                                            }
                                            else{
                                                if(month.equals("December")){
                                                    return 12;
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
//end if-else
return 0;
}
/**
 * function which take a month and returns a string

```



```

* @param int
* @return String
*/
String getmonthbyint(int month){
    if(month==1){
        String s="January";
        return s;
    }
    else{
        if(month==2){
            String s="February";
            return s;
        }
        else{
            if(month==3){
                String s="March";
                return s;
            }
            else{
                if(month==4){
                    String s="April";
                    return s;
                }
                else{
                    if(month==5){
                        String s="May";
                        return s;
                    }
                    else{
                        if(month==6){
                            String s="June";
                            return s;
                        }
                        else{
                            if(month==7){
                                String s="July";
                                return s;
                            }
                            else{
                                if(month==8){
                                    String s="August";
                                    return s;
                                }
                                else{
                                    if(month==9){
                                        String s="September";
                                        return s;
                                    }
                                    else{
                                        if(month==10){
                                            String s="October";
                                            return s;
                                        }
                                        else{
                                            if(month==11){
                                                String s="November";
                                                return s;
                                            }
                                            else{
                                                if(month==12){
                                                    String s="December";
                                                    return s;
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    }
    }
    }
    }
    return null;

} //end

/**
 * sort jobs by priority method
 */
void sortbypriority(){

    StepContent max, tmp;
    for(int i=0; i<job_queue.size(); i++){
        max=(StepContent) job_queue.elementAt(i);
        for(int j=i+1; j<job_queue.size(); j++){
            StepContent step1=(StepContent) job_queue.elementAt(j);
            if(max.getPriority()<step1.getPriority()){
                tmp=max;
                max=step1;
                job_queue.setElementAt(tmp, j);
            } //end if()
        } //end for(j)

        job_queue.setElementAt(max, i);
    } //end for(i)

} //end sort

/**
 * sort jobs by deadline method
 */
void sortbydeadline(){

    StepContent earlist, tmp;
    int y1=0, y2=0, m1=0, m2=0, d1=0, d2=0;
    for(int i=0; i<job_queue.size(); i++){
        earlist=(StepContent) job_queue.elementAt(i);

        String dead1=earlist.getDeadline();

        getdate(dead1);
        y1=0; m1=0; d1=0;
        y1=this.year; m1=this.month; d1=this.date;

        for(int j=i+1; j<job_queue.size(); j++){
            StepContent step1=(StepContent) job_queue.elementAt(j);
            String dead2=step1.getDeadline();
            getdate(dead2);
            y2=0; m2=0; d2=0;
            y2=this.year; m2=this.month; d2=this.date;

            int n=after(y1,m1,d1,y2,m2,d2);

            if(n==1){
                dead1=step1.getDeadline();
                getdate(dead1);
                y1=0; m1=0; d1=0;
                y1=this.year; m1=this.month; d1=this.date;
                tmp=earlist;
                earlist=step1;
                job_queue.setElementAt(tmp, j);
            } //end if()
        } //end for(j)
    }
}

```

```

        job_queue.setElementAt(earlist, i);
    } //end for(//i)

}

int after(int y1, int m1, int d1, int y2, int m2, int d2){
    if(y1>y2){
        return 1;
    }
    else{
        if(y1==y2){
            if(m1>m2){
                return 1;
            }
            else{
                if(m1==m2){
                    if(d1>d2){
                        return 1;
                    }
                }
            }
        }
    }
    return 0;
} //end after

void getdate(String date){
    try{
        this.year=0; this.month=0; this.date=0;
        StringTokenizer st = new StringTokenizer(date," ");
        st.hasMoreTokens();
        String s=new String(st.nextToken());

        this.month=getmonthbystring(s);
        String d=st.nextToken();
        this.year=Integer.parseInt(st.nextToken());

        StringTokenizer st1 = new StringTokenizer(d,"");
        st1.hasMoreTokens();
        String d2=new String(st1.nextToken());
        this.date=Integer.parseInt(d2);
    }
    catch(Exception e){}
}

/**
 * sort jobs by duration
 */
void sortbyduration(){
    StepContent min, tmp;
    for(int i=0; i<job_queue.size(); i++){
        min=(StepContent) job_queue.elementAt(i);
        for(int j=i+1; j<job_queue.size(); j++){
            StepContent step1=(StepContent) job_queue.elementAt(j);
            if(min.getDuration()>step1.getDuration()){
                tmp=min;
                min=step1;
                job_queue.setElementAt(tmp, j);
            } //end if()
        } //end for(j)

        job_queue.setElementAt(min, i);
    } //end for(//i)

}

/**

```

```

* sort jobs by start time
*/
void sortbystarttime(){
    int y1, y2, m1, m2, d1, d2;
    StepContent ealist, tmp;
    for(int i=0; i<job_queue.size(); i++){
        ealist=(StepContent) job_queue.elementAt(i);
        String dead1=ealist.getStartTime();
        getdate(dead1);
        y1=this.year; m1=this.month; d1=this.date;

        for(int j=i+1; j<job_queue.size(); j++){
            StepContent step1=(StepContent) job_queue.elementAt(j);
            String dead2=step1.getStartTime();
            getdate(dead2);
            y2=0; m2=0; d2=0;
            y2=this.year; m2=this.month; d2=this.date;

            if(after(y1,m1,d1,y2,m2,d2)==1){
                dead1=step1.getStartTime();
                getdate(dead1);
                y1=this.year; m1=this.month; d1=this.date;
                tmp=ealist;
                ealist=step1;
                job_queue.setElementAt(tmp, j);
            }
        }
        job_queue.setElementAt(ealist, i);
    }

}

/**
* sort jobs by laxity
*/
void sortbylaxity(){
    StepContent min, tmp;
    int laxity1,laxity2;
    int y1, y2, m1, m2, d1, d2;
    for(int i=0; i<job_queue.size(); i++){
        min=(StepContent) job_queue.elementAt(i);
        String deadline1=min.getDeadline();
        getdate(deadline1);
        y1=0; m1=0; d1=0;
        y1=this.year; m1=this.month; d1=this.date;

        String starttime1=min.getStartTime();
        getdate(starttime1);
        y2=0; m2=0; d2=0;
        y2=this.year; m2=this.month; d2=this.date;
        laxity1=((y1-y2)*360+(m1-m2)*30+(d1-d2))-min.getDuration();

        for(int j=i+1; j<job_queue.size(); j++){
            StepContent step=(StepContent) job_queue.elementAt(j);
            String deadline2=step.getDeadline();
            getdate(deadline2);
            y1=0; m1=0; d1=0;
            y1=this.year; m1=this.month; d1=this.date;
            String starttime2=step.getStartTime();
            getdate(starttime2);
            y2=0; m2=0; d2=0;
            y2=this.year; m2=this.month; d2=this.date;
            laxity2=((y1-y2)*360+(m1-m2)*30+(d1-d2))-step.getDuration();

            if(laxity1>laxity2){

```

```

        tmp=min;
        min=step;
        job_queue.setElementAt(tmp, j);
    }//end if()
} //end for(j)
job_queue.setElementAt(min, i);
} //end for(i)

} //end sortbylaxity()

//sortbyMinD_MinE
void sortbyD_E(){

    sortbydeadline();
    w=weight.get_w();
    StepContent step=(StepContent) job_queue.lastElement();
    String s=step.getDeadline();
    getdate(s);
    int y=year;
    int m=month;
    int d=date;

    StepContent max, tmp;

    for(int i=0; i<job_queue.size(); i++){
        max=(StepContent) job_queue.elementAt(i);
        String deadline1=max.getDeadline();
        getdate(deadline1);
        int y_dead1=this.year;
        int m_dead1=this.month;
        int d_dead1=this.date;
        for(int j=i+1; j<job_queue.size(); j++){
            StepContent step1=(StepContent) job_queue.elementAt(j);
            String deadline2=step1.getDeadline();
            getdate(deadline2);
            int y_dead2=year;
            int m_dead2=month;
            int d_dead2=date;
            if((((y-y_dead1)*360+(m-m_dead1)*30+(d-d_dead1))*w
                -max.getDuration()*(1-w))<
                (((y-y_dead2)*360+(m-m_dead2)*30+(d-d_dead2))*w
                -step1.getDuration()*(1-w)))
            {
                tmp=max;
                max=step1;
                job_queue.setElementAt(tmp, j);
            }
        } //end if()
    } //end for(j)

    job_queue.setElementAt(max, i);
} //end for(i)
}

void sortbyD_S(){

    sortbydeadline();

    w=weight.get_w();
    StepContent step=(StepContent) job_queue.lastElement();
    //int y=job.get_deadline().getYear();
    String s1=step.getDeadline();
    getdate(s1);
    int y_dead=year;
    int m_dead=month;
    int d_dead=date;

```

```

String s2=step.getStartTime();
getdate(s2);
int y_start=year;
int m_start=month;
int d_start=date;

StepContent max, tmp;
for(int i=0; i<job_queue.size(); i++){
    max=(StepContent) job_queue.elementAt(i);

    String s3=max.getDeadline();
    getdate(s3);
    int y_dead_max=year;
    int m_dead_max=month;
    int d_dead_max=date;

    String s4=max.getStartTime();
    getdate(s4);
    int y_start_max=year;
    int m_start_max=month;
    int d_start_max=date;

    for(int j=i+1; j<job_queue.size(); j++){
        StepContent step1=(StepContent) job_queue.elementAt(j);

        String s5=step1.getDeadline();
        getdate(s5);
        int y_dead_step=year;
        int m_dead_step=month;
        int d_dead_step=date;

        String s6=step1.getStartTime();
        getdate(s6);
        int y_start_step=year;
        int m_start_step=month;
        int d_start_step=date;

        if((((y_dead-y_dead_max)*365+(m_dead-m_dead_max)*30+(d_dead-d_dead_max))*w
            +((y_start-y_start_max)*365+(m_start-m_start_max)*30+(d_start-d_start_max))*(1-w))<
            (((y_dead-y_dead_step)*365+(m_dead-m_dead_step)*30+(d_dead-d_dead_step))*w
            +((y_start-y_start_step)*365+(m_start-m_start_step)*30+(d_start-d_start_step))*(1-w)))
        {
            tmp=max;
            max=step1;
            job_queue.setElementAt(tmp, j);

        }
    }
}

/**
 * method to handle standard action events
 */
public void actionPerformed(java.awt.event.ActionEvent event) {
    Object object = event.getSource();
    if (object == JButton5)
        JButton5_actionPerformed(event);
}

/**
 * handle JButton5 action event
 */
void JButton5_actionPerformed(java.awt.event.ActionEvent event){

```

```

        // to do: code goes here.
        this.setVisible(false);
    }

    // some standard focus events
    public void focusLost(FocusEvent event) { };
    public void focusGained(FocusEvent event) {
        Object object = event.getSource();
        if (object == JScrollPane2)
            JScrollPane2_focusGained(event);
        else if (object == JTable1)
            JTable1_focusGained(event);
    }

    /**
    * a method to handle standard item state changes for JComboBox
    */
    public void itemStateChanged(java.awt.event.ItemEvent event) {
        Object object = event.getSource();
        if (object == JComboBox1)
            JComboBox1_itemStateChanged(event);
    }

    /**
    * handle JComboBox item state changes
    */
    void JComboBox1_itemStateChanged(java.awt.event.ItemEvent event) {
        if( event.getStateChange() == event.SELECTED ){
            String str=(String) event.getItem();

            if(str.equals("High Priority First")){
                //new Getdat();
                sortBypriority();
                datavalu();
            }
            else{
                if(str.equals("Minimum Deadline First (Min_D)")){
                    sortBydeadline();
                    datavalu();
                }
                else
                    if(str.equals("Minimum Estimated Duration First (Min_E)")){
                        sortByduration();
                        datavalu();
                    }
                else{
                    if(str.equals("Minimum Earliest Start Time First (Min_S)")){
                        sortBystarttime();
                        datavalu();
                    }
                    else{
                        if(str.equals("Minimum Laxity First (Min_L)")){
                            sortBylaxity();
                            datavalu();
                        }
                        else{
                            if(str.equals("Min_D*w + Min_E*(1-w)")){
                                (new JDialog_weit(this, weight, 1)).setVisible(true);
                                //sortByD_E();
                            }
                            else{
                                if(str.equals("Min_D*w + Min_S*(1-w)")){
                                    (new JDialog_weit(this, weight, 2)).setVisible(true);
                                    //sortByD_S();
                                }
                            }
                        }
                    }
                }
            }
        }
    }

```

```

        }
    }
}

}

}

/**
 * handle focus event
 */
void JScrollPane2_focusGained(java.awt.event.FocusEvent event) {
    // to do: code goes here.
}

/**
 * handle focus event
 */
void JTable1_focusGained(java.awt.event.FocusEvent event) {
    // to do: code goes here.
}
}

```

9. JobSchedule.Predecessor

```

/**-----
 * Filename: Predecessor.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Compiler: JDK 1.3.1
 * Description: A predecessor class.
 *-----
 */
package Cases.JobSchedule;

import java.io.Serializable;

import java.util.Vector;

public class Predecessor implements Serializable{
    private Vector predec;
    public Predecessor(){predec=new Vector();}
    public Vector get_predec(){ return predec;}
}

```

10. JobSchedule.Result

```

/**-----
 * Filename: Result.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Compiler: JDK 1.3.1
 * Description: a class which holds the id and grade of a person.
 *-----
 */
package Cases.JobSchedule;

import java.io.Serializable;

public class Result implements Serializable{

```



```

private int personid;
private int grade;
public Result(int id, int grade1){
    personid=id;
    grade=grade1;
}

public Result() {}

public int get_id(){ return personid; }
public int get_grade(){ return grade; }
}

```

11. JobSchedule.Weight

```

/**-----
 * Filename: Weight.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Compiler: JDK 1.3.1
 * Description: A weight class.
 *-----
 */
package Cases.JobSchedule;

import java.io.Serializable;

public class Weight implements Serializable{
    private double w;
    private int i;
    public Weight(){ w=0.5; i=5;}
    public Weight(double w1){
        w=w1;
    }
    public void put_w(double w1){w=w1;}
    public void put_i(int i1){i=i1;}
    public double get_w(){return w;}
    public int get_i(){return i;}
}

```

12. JobSchedule.Work_schedul

```

/**-----
 * Filename: Work_Schedul.java
 * @Date: 2-20-2003
 * @Author: Hahn Le
 * Compiler: JDK 1.3.1 (needs implementation)
 * Description: A Work_Schedul class.
 *-----
 */
package Cases.JobSchedule;

import java.io.Serializable;

import java.util.Vector;

public class Work_schedul implements Serializable{
    private int personid;
    private Vector schedul;
    public Work_schedul(int id){
        personid=id;
        schedul=new Vector();
    }
}
/**

```

```
* default constructor
*/
public Work_schedul() {}
public int get_id(){ return personid;}
public Vector get_schedul(){ return schedul;}
}
```

LIST OF REFERENCES

- [AKAO90] Akao, Y., *Quality Function Deployment: Integrating Customer Requirements into Product Design*, Tamagawa University, Japan translated by Glenn H. Mazur and Japan Business Consultants, Ltd., Productivity Press, Cambridge, MA, 1990.
- [COHE95] Cohen, L., *Quality Function Deployment: How to Make QFD Work for You*, Addison-Wesley, 1995.
- [CONK88] Conklin J. and Begeman M., "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," *ACM Transactions on Office Information System*, Vol. 6, October 1988, pp. 303-331.
- [CLAU88] Clausing, D., "Quality Function Deployment," in Ryan, N. E., ed., *Taguchi Methods and QFD*, American Supplier Institute Inc., Dearborn, MI, 1988.
- [HARN99a] Harn, M., "Computer-Aided Software Evolution Based on Inferred Dependencies" *Ph.D. Dissertation*, Computer Science Department, Naval Postgraduate School, Monterey, CA, 1999.
- [HARN99b] Harn, M., Berzins, V., and Luqi, "Software Evolution Process via a Relational Hypergraph Model," *Proceedings of IEEE/IEEEJ/JSAP International Conference on Intelligent Transportation Systems*, Tokyo, Japan, October 5-8, 1999, pp. 599-604.
- [HAUS88] Hauser, J. R. and Clausing, D., "The House of Quality," *The Harvard Business Review*, May-June 1988, No. 3, pp 63-73.
- [LEHC99] Le, Hahn C., "Design of a Persistence Server for the Relational Hypergraph Model", Masters Thesis, Computer Science Department, Naval Postgraduate School, Monterey, CA 1999.
- [PRES01] Pressman, R., "Software Engineering: A Practitioner's Approach", Fifth Edition, McGraw Hill, 2001.
- [PUET02] Puett, J., "Holistic Framework for Establishing Interoperability of Heterogeneous Software Development Tools and Models," *Proc. 24th Intl. Conf. on Software Engr.*, Orlando FL, May 2002, pp. 729-730.

- [PUET03] Puett, J., "Holistic Framework for Establishing Interoperability of Heterogeneous Software Development Tools," *PhD Dissertation*, Computer Science Department, Naval Postgraduate School, Monterey CA, June 2003.
- [RATI01] Rational Software Corporation, "Rational ProjectConsole Getting Started", *Manual, Version: 2002.05.00, Part Number: 800-025142-000*, 2002.
- [ZULT90] Zultner, R. E., "Software Quality Deployment: Adapting QFD to Software," *Transactions of the Second Symposium on Quality Function Deployment*, Novi MI, 18-19 June 1990, pp. 132-149.
- [ZULT92] Zultner, R. E., "Quality Function Deployment (QFD) for Software: Structured Requirements Exploration," in Schulmeyer, G. G. and J. I. McManus, ed., *Total Quality Management for Software*, Van Nostrand Reinhold, New York NY, 1992.
- [ZULT93] Zultner, R. E., "TQM for Technical Teams," *Communications of the ACM*, Vol. 36, No. 10, 1993, pp. 79-91.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center
Ft. Belvoir, Virginia

Dudley Knox Library
Naval Postgraduate School
Monterey, California

Professor Peter J. Denning
Naval Postgraduate School
Monterey, California

Professor Man-Tak Shing
Naval Postgraduate School
Monterey, California

LTC Joseph F. Puett III
Naval Postgraduate School
Monterey, California

Professor Richard D. Riehle
Naval Postgraduate School
Monterey, California

Mr. Douglas Lange
Space and Naval Warfare Systems Center
San Diego, California

Dr. David Hislop
U.S. Army Research Office
Research Triangle Park, North Carolina