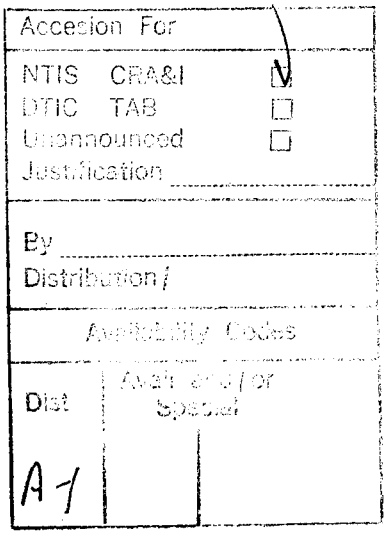
Using Tools in a Leveraged Synthesis Process

SPC-94067-CMC
Version 01.00.01

December 1994



The tools mentioned in this report are not specifically recommended or evaluated, either among themselves or relative to other commercially available tools that are not mentioned. Furthermore, the accuracy and completeness of individual tool descriptions have not been validated with the vendors offering these tools and should not be considered authoritative. For such information, the respective vendors should be contacted directly.

CONTENTS

ACKNOWLEDGMENTS ix

EXECUTIVE SUMMARY xi

1. INTRODUCTION 1

1.1 Overview 1

1.2 Document Audience and Purpose 1

1.3 Document Structure 1

1.4 Typographic Conventions 2

2. NEEDS FOR METHODS AND TOOLS 3

2.1 Method Needs 3

2.2 Tool Needs 3

2.2.1 Domain Management 4

2.2.2 Domain Analysis 4

2.2.3 Domain Implementation 4

2.2.4 Needs for Project Support 4

2.2.5 Application Engineering 5

3. TYPICAL TOOLING SCENARIOS 7

3.1 Rockwell CCSD Pilot Project Scenario 7

3.1.1 Tool Usage Summary 7

3.1.2 Process Artifact and Tool Walkthrough 9

3.1.2.1 Domain Analysis 9

3.1.2.2 Domain Implementation 11

3.1.2.3 Application Engineering 14
3.1.3 Tooling Improvement Opportunities 14
3.2 Boeing/Navy STARS Demonstration Project 16
3.2.1 Tool Usage Summary 17
3.2.2 Process Artifact and Tool Walkthrough 20
3.2.2.1 Domain Analysis 20
3.2.2.2 Domain Implementation 20
3.2.2.3 Application Engineering 21
3.2.3 Opportunities for Improvement 28
3.3 Case Studies Analysis 28
3.3.1 Methods 28
3.3.2 Tools 29
4. ENHANCED TOOL SUPPORT 31
4.1 A Typical Tool Base 31
4.2 Tools for Domain Management 32
4.2.1 Tools for Domain Planning 32
4.2.2 Tools for Process Management 32
4.2.3 Tools for Configuration Management 33
4.3 Tools for Domain Definition 34
4.4 Tools for Domain Analysis 35
4.4.1 Tools for Decision Modeling 35
4.4.2 Tools for Product Requirements and Design 35
4.4.3 Tools for Process Requirements 36
4.5 Tools for Domain Implementation 36
4.5.1 Tools for Product Implementation 36
4.5.1.1 Documentation Tools 36
4.5.1.2 Code Implementation Tools 37

4.5.1.3 User Interface Builder Tools 37

4.5.1.4 Testing Tools 38

4.5.2 Tools for Process Support Development 38

4.5.2.1 Tools for Representing an Application Model 38

4.5.2.2 Tools for Providing a User Interface for Application Modeling 40

4.5.2.3 Domain-Specific Tools for Application Engineering 40

4.6 Tools for Project Support 41

5. CONCLUSIONS 43

APPENDIX A. BOEING/STARS SEE EVALUATION 45

APPENDIX B. USING FILEMAKER PRO TO REPRESENT DECISION AND APPLICATION MODELS 49

B.1 Operations Provided by FileMaker Pro 49

B.2 Using FileMaker Pro During the Decision Model Activity 50

B.3 Using FileMaker Pro During the Process Requirements Activity 51

B.4 Using FileMaker Pro During the Process Support Development Activity 51

B.5 Using FileMaker Pro During the Application Engineering Activity 51

APPENDIX C. CREATING ADAPTABLE DOCUMENTATION 53

C.1 Interleaf Mechanisms 53

C.2 Microsoft Word Mechanism 53

C.3 WordPerfect Mechanisms 53

C.4 FrameMaker Mechanism 54

C.5 Layering Adaptability Onto an Interleaf Document With TRF2 54

C.6 Comparison of Approaches 54

APPENDIX D. AN INFORMATION MODEL FOR LEVERAGED SYNTHESIS 57

D.1 Domain Engineering Process 58

D.2 Application Engineering Process 59

D.3 Domain 60

D.4 Domain Plan 61

D.5 Domain Definition 62

D.6 Decision Model 63

D.7 Product Family Requirements 64

D.8 Product Family Design 65

D.9 Product Family Implementation 66

D.10 Process Requirements 67

D.11 Process Support 68

APPENDIX E. REUSE LIBRARY TOOLS 69

E.1 Reuse Library Tools 69

E.2 Typical Reuse Library Tool Capabilities 70

E.3 Using Reuse Library Tools in Synthesis 71

E.3.1 Reuse Library Tool Use in Application Engineering 71

E.3.2 Reuse Library Tool Use in Domain Engineering 71

APPENDIX F. TOOL CONTACT INFORMATION 73

LIST OF ABBREVIATIONS AND ACRONYMS 79

GLOSSARY 81

REFERENCES 85

FIGURES

Figure 1. Adaptable RTSA Specification Fragment 11

Figure 2. Bus Controller Software Architecture DDC Fragment 12

Figure 3. Bus Controller Software Architecture DTI Fragment 12

Figure 4. An Adaptable Requirements Component 13

Figure 5. WordPerfect Macro Fragments for Generating an SRS 14

Figure 6. A Screen From the Application Engineering Environment 14

Figure 7. A Generated Requirements Document Component 15

Figure 8. A KAMEL MAKE Function 22

Figure 9. A KAMEL Decide Screen 23

Figure 10. AAA (Denali) Process Enactment Code 24

Figure 11. Resulting Charge Line Query Dialog Window 24

Figure 12. KAMEL-Generated Retrieve/Adapt Directives (Excerpt) 26

Figure 13. Interleaf ASCII Generated From Retrieve/Adapt Directives (Excerpt) 27

Figure 14. Final Generated Interleaf System/Segment Specification (Excerpt) 27

TABLES

Table 1. Rockwell CCSD Tool Usage Summary 8

Table 2. A Decision Class From the System Bus Decision Model 10

Table 3. Navy/STARS Tool Usage Summary 17

Table 4. SEE Infrastructure Summary 23

Table 5. Summary of How FileMaker Pro Supports Synthesis 49

ACKNOWLEDGMENTS

Rich McCabe was the project manager and a contributor to this report. Other authors were Grady Campbell, Ted Davis, Jeff Facemire, and Steve Wartik. Reviewers were Mark Blackburn, Mark Tappan, and Steve Wartik. The efforts of the Rockwell and Boeing pilot projects were essential to our understanding of how commercial tools can support a leveraged Synthesis process.

This page intentionally left blank.

EXECUTIVE SUMMARY

The leveraged Synthesis process, described in the *Reuse-Driven Software Processes Guidebook*, has been in use by Rockwell and Boeing, including the ARPA-sponsored Software Technology for Adaptable Reliable Systems (STARS) Demonstration Project with the Navy, since 1991. Although not required for successful use of a Synthesis process, appropriate tools can make the process less expensive and more effective.

This report analyzes the differing approaches to automation used by the Rockwell and Boeing leveraged Synthesis pilot projects:

- Rockwell chose a limited-capability, low-cost approach, making use of commercial off-the-shelf tools.
- Boeing undertook, as a major task of their project under the STARS program, to create an extensive automated environment that would support their use of the process.

Both of these approaches have been successful in providing effective support for performing the process. These two experiences are portrayed as prototypical scenarios that future adopters of leveraged Synthesis might viably emulate. The analyses describe which activities the pilot projects augmented with tools and how they used those tools.

As a foundation for understanding alternative approaches to tooling, a brief analysis of process-based needs for methods and tools is provided. Based on pilot project and other experiences, this report surveys the applicability of various types of commercially available tools in support of those needs. Analysis of the pilot projects revealed two key points:

- Adopting a Synthesis process stimulates the adoption of standardized methods, particularly for requirements and design.
- A pilot project can show the potential attainable with a Synthesis process without substantial investments in tooling. However, realization of that potential requires commitment to use the process by a business organization and standardization on preferred tools.

In conclusion, this report makes recommendations on how adopters should approach tooling and how commercial tool vendors might provide more effective support for a leveraged Synthesis process. Principal among these recommendations are to:

- Focus initially on the tooling needs of Application Engineering, which offers the greatest opportunity for near-term payoff, and begin with readily available tools.
- Invest in tool enhancements that support creation and use of adaptable work products.

- Create project management tools that enable an iterative process, particularly emphasizing milestones based on work product versioning.

The appendixes provide analyses of various tool-related questions:

- The Boeing environment is seen to be unique in providing support specifically oriented to a leveraged Synthesis process. However, at this early stage in its development, work remains before it fully supports the envisioned process.

- The FileMaker Pro database tool is seen to be useful, though greatly limited, as an enabling mechanism for Application Modeling.

- Various document processors (FrameMaker, Interleaf, Microsoft Word, and WordPerfect) are seen to provide capabilities for creating adaptable documentation and tailored instances. Their capabilities are compared, along with an approach of embedded use of the TRF2 metaprogramming capability with a document processor tool.

- Reuse library tools are shown to have a role within a leveraged Synthesis process but a much less visible one than conventionally expected.

In addition, an information model of Synthesis concepts is provided as a foundation for future tool development efforts.

1. INTRODUCTION

1.1 OVERVIEW

This report is an analysis and survey of the needs and alternatives for automated support of a leveraged Synthesis process as documented in the *Reuse-Driven Software Processes Guidebook* (RSP Guidebook) (Software Productivity Consortium 1993). It describes automation needs relative to each of the major activities of the process. Alternatives for automation are identified based on the capabilities of generally accessible commercial tools. As a practical guide to how tools can be used, this report describes previous experiences in the use of tools for two pilot projects: Rockwell Command and Control Systems Division (CCSD) and the Boeing/Navy Software Technology for Adaptable Reliable Systems (STARS) project. The appendixes provide detailed, but informal, analyses on certain uses of particular tools, chosen to illustrate the potential for automated support of a Synthesis process.

1.2 DOCUMENT AUDIENCE AND PURPOSE

The purpose of this report is to identify various current opportunities for more effective practice of a leveraged Synthesis process through the attendant adoption of automated tools. This report is targeted primarily to organizations that have begun to adopt a leveraged Synthesis process and wish to enhance their practice with automated tools. This report is targeted secondarily to vendors of commercial tools to understand how tools might be designed to support a leveraged Synthesis process more effectively. Readers of this report are assumed to be familiar with the leveraged Synthesis process as defined in the guidebook.

1.3 DOCUMENT STRUCTURE

This report discusses how automation can support a leveraged Synthesis process. It is organized as follows:

- Section 2 defines informal requirements for the automation of a Synthesis process, including needs for both methods and tools.
- Section 3 describes and compares the experiences with automated support for the leveraged Synthesis process of the Rockwell CCSD pilot project and the Boeing/Navy STARS demonstration project.
- Section 4 identifies alternatives for automated support, with commercially supported tools, of the various activities of a leveraged Synthesis process.
- Section 5 summarizes the state of tool support for a leveraged Synthesis process and identifies potential improvements that tool vendors might pursue.

- Appendix A is an informal evaluation of the Boeing Software Engineering Environment (SEE) for use as a leveraged Synthesis Application Engineering Environment.
- Appendix B is an informal evaluation of the FileMaker Pro tool as a mechanism for automated support of Decision Modeling and Application Modeling.
- Appendix C is an analysis of how organizations can create adaptable documentation using commercial document processors.
- Appendix D is a draft information model for automated support of a leveraged Synthesis process.
- Appendix E is a discussion of reuse library tools and how they are used within a leveraged Synthesis process.
- Appendix F is a listing of all the tools mentioned in this report and contact information for their respective vendors.

1.4 TYPOGRAPHIC CONVENTIONS

This report uses the following typographic conventions:

Serif font ..... General presentation of information.

Italicized serif font ..... Publication titles.

Boldfaced serif font ..... Section headings and emphasis.

Boldfaced italicized serif font ..... Run-in headings in bulleted lists.

2. NEEDS FOR METHODS AND TOOLS

This section identifies the needs associated with a leveraged Synthesis process for software engineering methods and tools. To institute such a process, an organization must choose a set of software methods to tailor the process definition into a practicable process. In addition, they can select supporting tools and further tailor the process definition to make performance of the process easier or more efficient.

2.1 METHOD NEEDS

To be complete, a leveraged Synthesis process must be elaborated with an organization's preferred methods for:

- Management (in Domain Management)
- Requirements analysis and specification (in Product Requirements and Process Requirements)
- Design and implementation (in Product Design, Product Implementation, and Process Support Development)
- Verification (in Domain Implementation and Domain Verification).

Examples of such methods, respectively, are the Consortium's Evolutionary Spiral Process (ESP) method, the Consortium Requirements Engineering (CoRE) method, the Consortium's Ada-based Design Approach for Real-Time Systems (ADARTS℠) method, Ada Quality and Style, and the Cleanroom correctness verification method (Hausler, Linger, and Trammell 1994). There are numerous other equivalent methods, including process-oriented (e.g., structured), data-oriented (e.g., information engineering), and object-oriented methods. Alternatively, even commonly practiced ad hoc methods suffice from a process perspective as long as their use is standardized by the organization for a domain. Within the context of any particular domain, one method of each type must be chosen as the standard to be followed. The set of methods chosen must be sufficiently compatible to allow integrated use and information traceability among the work products of the methods.

2.2 TOOL NEEDS

Although a leveraged Synthesis process can be performed entirely manually, there is obvious benefit in augmenting and supporting it with appropriate automation. Automation can be custom-built, commercial off-the-shelf, or a combination. Because no commercially supported tools have yet been designed specifically to support family-based development, their application must be considered with care and introduced in ways that do not conflict with the needs of Synthesis activities. However, the

utility of commercial tools must be optimally exploited to minimize the need for investment in custom tool development by organizations for which tool development is not a business objective in its own right.

2.2.1 Domain Management

Domain Management requires tools for planning, monitoring, and controlling the activities of the Domain Engineering effort and its comprising increments. This includes capabilities to define tasks with associated objectives, schedule the tasks, assign them to individuals or teams, and track progress against a plan. In addition, tools for configuration management support controlling the versions of the Domain Engineering product as a set of evolving work products. Tools for quality assurance support controlling the quality of the Domain Engineering process in practice and of its product.

2.2.2 Domain Analysis

Domain Analysis requires tools for:

- Organizing information at both a conceptual and a formal semantic level, including definitions of terms and relationships
- Specifying the requirements and design of the systems encompassed by a domain
- Specifying the process of Application Engineering within the domain and the forms by which information describing any particular system is presented and analyzed
- Verifying the content and quality of Domain Engineering work products

2.2.3 Domain Implementation

Domain Implementation requires tools for:

- Implementing and verifying adaptable forms of application work product components
- Implementing and verifying mechanisms for the instantiation and composition of adaptable work product components into complete instance work products
- Implementing the Application Engineering process specified by Domain Analysis
- Documenting the Application Engineering process and supporting automation for application engineers
- Supporting installation of automation in project environments

2.2.4 Needs for Project Support

Project Support primarily uses the tools provided by Domain Implementation to validate Application Engineering Process Support (the Domain Engineering product) and deliver it to client application engineering projects. The only additional tool needed is for the recording of problems and needed improvements discovered as the Application Engineering process is performed.

2.2.5 Application Engineering

Application Engineering uses the tools provided by Domain Engineering to produce an Application Product for a customer. Decisions on appropriate tooling are the responsibility of Domain Implementation, as part of the Process Support Development activity.

This page intentionally left blank.

3. TYPICAL TOOLING SCENARIOS

This section describes two major efforts using the leveraged Synthesis process: Rockwell's communications control and management systems projects and the Boeing/STARS/Navy training systems Demonstration Project. This section includes a summary of each effort relating to the methods and/or tools used to support the Synthesis activities. It includes selected artifacts illustrating the two approaches. This section identifies opportunities for improved tooling, and it closes with an analysis and comparison of the situational factors that led to differences in how Rockwell and Boeing STARS applied Synthesis.

3.1 ROCKWELL CCSD PILOT PROJECT SCENARIO

In December 1990, the CCSD of Rockwell initiated a Consortium-assisted pilot project to evaluate the applicability of the leveraged Synthesis process to Rockwell's software engineering needs. CCSD application of a Synthesis process has resulted in the production of a partially automated environment that supports the specification of a system in the targeted CCSD domain and the generation of corresponding software requirements, design, and code.

A domain-specific notation created by the project lets an engineer describe a system in terms of high-level requirements and engineering decisions. A corresponding product is generated by mechanically selecting, adapting, and composing reusable components based on the decisions expressed in the specification. The utility of this environment has been demonstrated through successful creation of parts of two products and limited use on a current CCSD project.

CCSD has applied the Synthesis methodology to two domains. First, CCSD focused on the Communications Control and Management domain, specifically the software that supports communication over the MIL-STD-1553B system bus. During this effort, CCSD developed a tools approach for specifying and generating software work products. CCSD is now applying this same approach to their business area of Multimedia Message Handling Systems.

This section describes CCSD's use of tools in the application of the Synthesis methodology. O'Connor et al. (1994) describe CCSD's experience in applying Synthesis to the MIL-STD-1553B system bus domain.

3.1.1 Tool Usage Summary

Table 1 summarizes CCSD's use of tools in support of Synthesis activities. It indicates the methods and tools used and states how the indicated tools support the corresponding activity. References to "RSP" in the Method column mean that the project performed the activity following only the general guidance provided by the RSP Guidebook (Software Productivity Consortium 1993). References to "ad hoc" mean that the project did not fully apply RSP guidance for an activity.

Table 1. Rockwell CCSD Tool Usage Summary

Activity	Method	Tool	Tool Usage
Domain Management	RSP PERT	Timeline	Scheduling and tracking domain development activities.
Domain Definition	RSP	WordPerfect	Documenting the Domain Synopsis, Glossary, and commonality and variability assumptions.
Decision Model	RSP	Word Perfect GEM Draw	Documenting decisions to resolve variabilities in the form of tables and define decision variables.
Product Requirements	RTSA	OpenSelect	Developing data flow and control flow diagrams to specify Product Requirements. Annotated with decision variables to denote variations.
Process Requirements	Ad Hoc	NA	NA
Product Architecture	ADARTS	WordPerfect GEM Draw	Documenting and illustrating an ADARTS design for the product family. Annotated with decision variables to denote variations.
Component Design	PDL	WordPerfect GEM Draw	Documenting specifications for Adaptable Components. Parameterized with the decision variables.
Generation Design	Ad Hoc	NA	NA
Domain Verification	RSP	NA	NA
Component Implementation	RSP	WordPerfect Ada, C TRF2	Implementing adaptable documentation components with WordPerfect macro variables. Implementing adaptable Ada and C software components.
Generation Implementation	RSP	WordPerfect	Implementing a menu-based Application Modeling and generation environment for producing documentation and software work products using WordPerfect macros and merge capabilities.
Process Support Development	RSP	WordPerfect	Developing user documentation.
Project Support	RSP		
Application Engineering	RSP	WordPerfect Ada, C	Specifying applications and automatically generating work products.

As the table indicates, CCSD relied principally on Synthesis for process and method guidance. CCSD recognized the value of Synthesis as a disciplined engineering process; they discovered that attaining its full potential required acceptance of disciplined engineering methods as well. Synthesis does not dictate particular methods and, in fact, only requires (as a minimum) standardization of the way products are represented. However, Synthesis provides a framework within which organizations can adopt disciplined methods without having to retrain every engineer.

CCSD complemented the Synthesis methodology with Real-Time Structured Analysis (RTSA), ADARTS, and a Program Design Language (PDL). CCSD found these methods to be very useful in promoting reusability but noted that representing variation in graphical requirements and design methods was a very difficult problem. Section 3.1.2.1 illustrates how CCSD handled this problem.

CCSD focused their automation efforts on the Application Engineering process. Their motive was based on their experience that automated processes would be more readily accepted by the targeted users (i.e., in-house engineers) and would make them more receptive to the concepts of Synthesis. CCSD developed the Application Engineering environment through an innovative use of WordPerfect macros and merge capabilities on a Windows/PC platform. The environment enabled application engineers to interactively specify decisions to model an application and generate the application work products from Adaptable Components. Section 3.1.2.2 illustrates CCSD's use of WordPerfect macros.

CCSD's automated support for domain engineering was primarily used for documenting domain work products, such as the Domain Synopsis, Glossary, and Assumptions. Furthermore, in the case of product architecture design, CCSD used documentation tools to support ADARTS rather than tools that have semantic knowledge of the ADARTS notation, such as ObjectMaker. In addition, no automation was used to support configuration management. CCSD's use of OpenSelect for Product Requirements and WordPerfect for generation implementation are notable exceptions; CCSD could use these tools to analyze the work products as well as document them. Nonetheless, Domain Engineering is a strong candidate for further automation support.

3.1.2 Process Artifact and Tool Walkthrough

A central concept in a Synthesis process is the management of variation. How well an organization identifies, controls, and implements variabilities directly impacts the organization's reuse effectiveness. Dealing with the large number of legitimate variabilities that occur in a domain and their interrelationships can be a very complex and overwhelming task. Managing variation thus presents an opportunity and challenge for tool support.

Current off-the-shelf tools were not designed to accommodate variation. However, as CCSD demonstrated, off-the-shelf tools can provide sufficient support under certain circumstances with some improvising. This section illustrates how CCSD used tools to manage variation by walking through selected artifacts from their Domain Analysis, Domain Implementation, and Application Engineering activities, leading to the automatic generation of a Software Requirements Specification (SRS).

3.1.2.1 Domain Analysis

The objectives of Domain Analysis are to determine the scope of the domain, capture domain knowledge, and specify the product family and Application Engineering process. The products of Domain Analysis include the Domain Definition and Domain Specification.

Domain Definition. The Domain Definition is an informal description of systems in a business area and characterizes how the systems are similar and how they differ. CCSD developed a Domain Synopsis, Glossary, and Assumptions in accordance with the RSP Guidebook to make up the Domain Definition. These products are textual in nature, thus CCSD used WordPerfect as a documentation tool to record the Domain Definition.

The Domain Assumptions describe what is common, variable, and excluded among systems in the domain. This represents the beginning point for the management of variation. Here, commonalities and variabilities were identified in simple lists. A commonality from the system bus domain is:

> All systems communicating over the system bus will use the MIL-STD-1553B protocol.

Variabilities in the system bus domain include:

> Subsystems communicating over the system bus vary in their amount of RAM and ROM, type of processor, and 1553B chip set.

Domain Specification. The commonalities and variabilities, and other Domain Definition products, provide the basis for developing the Domain Specification. The Domain Specification is a precise specification of the problems and solutions supported by the domain, as well as the Application Engineering process for building systems in the domain. CCSD developed a decision model, Product Requirements, and Product Design in accordance with the RSP Guidebook to comprise the Domain Specification. CCSD did not formally develop Process Requirements.

Table 2 illustrates a portion of CCSD's decision model for the system bus domain. The decision model, which CCSD represented as a WordPerfect table, identifies the Application Engineering requirements and engineering decisions that determine how product family members vary. CCSD created the decision table by elaborating their variability assumptions; related decisions were grouped together in a decision class. Note that CCSD associated a decision variable (e.g., SSID, SSRAM, etc.) with each decision. CCSD used these decision variables throughout the process as a means to denote variations.

Table 2. A Decision Class From the System Bus Decision Model

Decisions	Value Space	Description
Subsystem Identifier[SSID]	identifier	Unique identifier for a 1553B subsystem
RAM[SSRAM]	hex [(0 .. ~)]	Amount of random access memory
ROM[SSROM]	hex [(0 .. ~)]	Amount of read only memory
Processor[SSPROC]	enum of (Intel80186, Intel80286, Intel80386)	Type of subsystem processor
1553B Chip Set [SSCHIP]	enum of (UT1553B BCRT, CT-1612, DDC BUS-65515, DTI-1121)	Type of MIL-STD-1553B hardware

CCSD developed the Product Requirements using OpenSelect to create a parameterized RTSA work product. The Product Requirements define the requirements for a product family and are adaptable to the decisions supported by the decision model. OpenSelect did have an inherent understanding of the RTSA method, but did not have an inherent ability to denote variations. CCSD compensated for

this by using OpenSelect's leveling capability to decompose RTSA data transformations (i.e., "bubbles") into multiple versions corresponding to alternative decision model choices. CCSD then annotated the data transformations with the decision variables to denote the variations. Figure 1 shows how the RTSA specification varies due to the SSCHIP decision being UT1553B BCRT, CT-1612 or DDC BUS-65515. It represents a decomposition of the Monitor Subsystem Hardware data transformation. In this case, CCSD used the SSCHIP decision variable to indicate alternative data transformations.

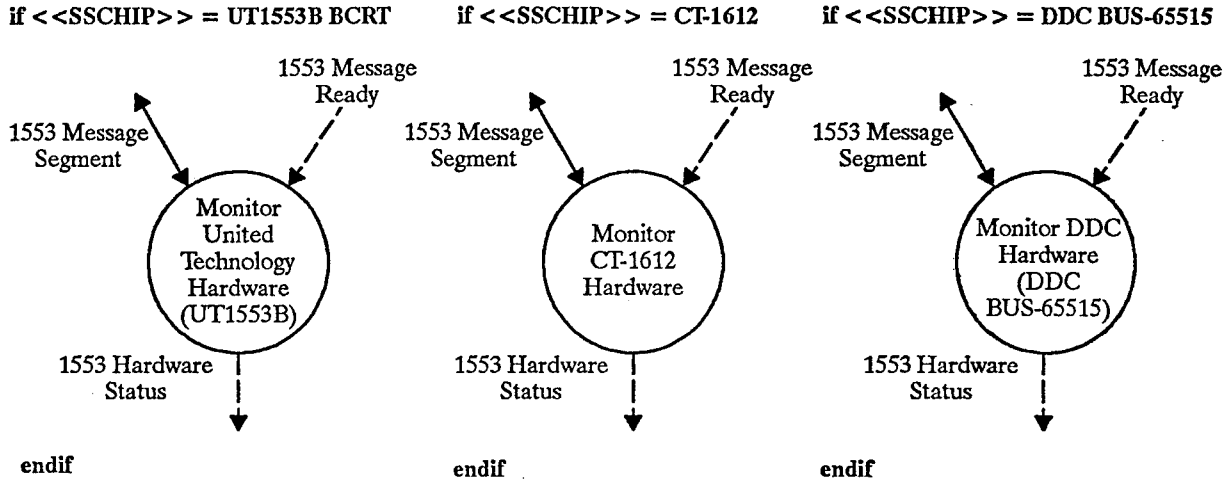


Figure 1. Adaptable RTSA Specification Fragment

The Product Design specifies the design for the product family, rather than for a single product. Like a single Product Design, however, the family Product Design must satisfy its Product Requirements. Like the Product Requirements, the Product Design must be adaptable to the decisions allowed by the family's decision model. The Product Design is composed of the product architecture, component design, and generation design.

CCSD developed the product architecture using the ADARTS methodology with GEM Draw. Unlike OpenSelect, GEM Draw did not understand the semantics of the method's notation. Similar to the Product Requirements, CCSD parameterized the design according to the decision model except that in this case CCSD used multiple diagrams to denote design variations. CCSD resorted to multiple diagrams rather than using a single, annotated diagram to show varying transformations as in Figure 1, because a single ADARTS diagram would have been overly complicated by the annotations. Figures 2 and 3, respectively, show how the architecture varies due to the SSCHIP decision being either DDC BUS-65515 or DTI-1121. This technique, although limited, worked because the architecture was relatively invariant with respect to decision Model Choices.

For the component design, CCSD represented variation in the specification of Adaptable Components using parameterized PDL. CCSD did not formally develop a generation design.

3.1.2.2 Domain Implementation

Domain Implementation is an activity of domain engineering for implementing product and process support for Application Engineering. The objectives of Domain Implementation are to create Adaptable Components and generation procedures as specified in the Product Design and to create a standardized Application Engineering process as specified in the Process Requirements.

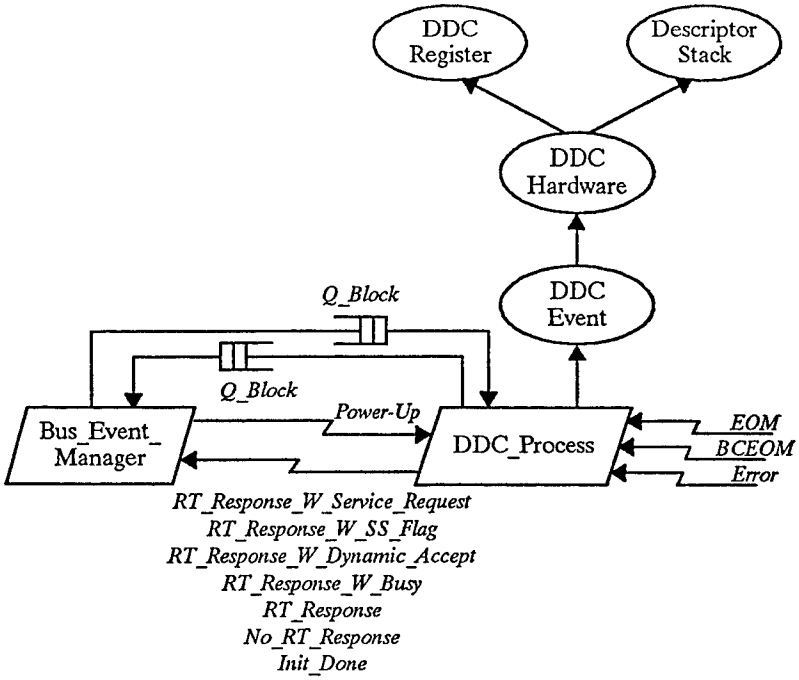


Figure 2. Bus Controller Software Architecture DDC Fragment

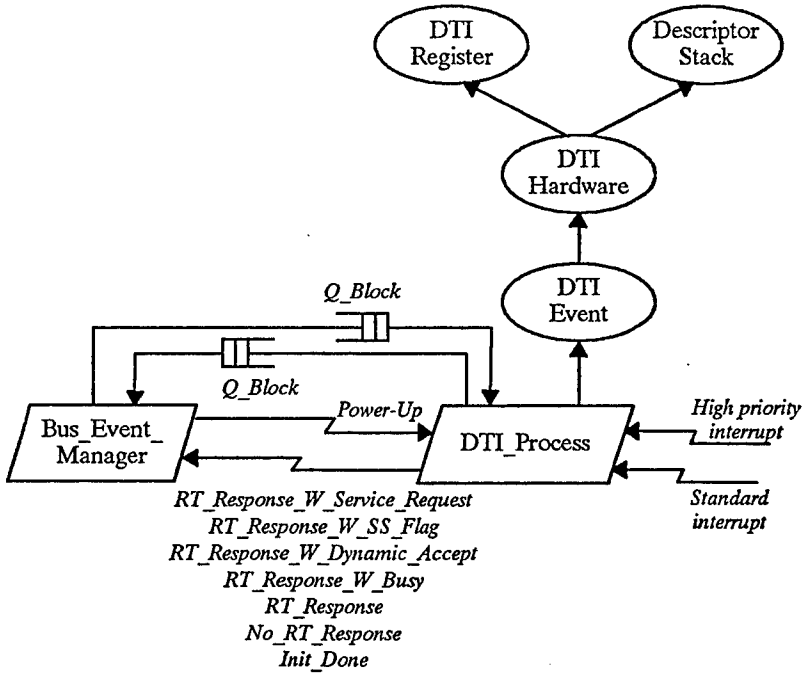


Figure 3. Bus Controller Software Architecture DTI Fragment

Adaptable Components. A component is any work product fragment (e.g., software, documentation, test scripts, etc.) produced during the Application Engineering process. CCSD developed adaptable requirements, design, and code components.

- ***Adaptable Requirements Components.*** These components are used to produce software requirements specifications for systems in the domain. The Product Requirements developed during the Domain Analysis determined the content of these components. Customer

requirements determined the form of the work products. CCSD used WordPerfect to create requirements components that could be automatically adapted based on Decision Model choices. WordPerfect's merge and macro features allow many document variations to be produced from a single template. Figure 4 shows a fragment of an adaptable requirements component represented using WordPerfect.

```
~a. {VARIABLE}ssid~ configuration. The {VARIABLE}appl~ application soft-
ware and the interprocessor communications software in the {VARIABLE}ssid~
subsystem will be stored in ROM and executed on an {COMMENT}
~{IF}{FIELD}ssproc~=1~Intel80186 {ENDIF}{COMMENT}
~{IF}{FIELD}ssproc~=2~Intel80286 {ENDIF}{COMMENT}
~{IF}{FIELD}ssproc~=3~Intel80386 {ENDIF}{COMMENT}
processor. The {VARIABLE}ssid~ subsystem contains {VARIABLE}ssrom~K
of ROM and {VARIABLE}ssram~K of RAM. The {VARIABLE}appl~ applica-
tion uses the MIL-STD-1553B bus to exchange messages with applications existing
in other subsystems in the network. The {VARIABLE}appl~ application commu-
nicates over the MIL-STD-1553B bus(es) via the terminal(s) described below.
```

Figure 4. An Adaptable Requirements Component

- *Adaptable Design Components.* The design components are used to produce software design specifications for systems in the domain. The content of these components was determined from the Product Design developed during the Domain Analysis. As with the adaptable requirements components, customer requirements determined the form of the work products, and CCSD implemented these components using WordPerfect.

- *Adaptable Code Components.* CCSD implemented code components in Ada, using interchangeable components (i.e., different implementations of a single interface), Ada generics, and the Consortium's TRF2 metaprogramming notation as mechanisms to represent a family of components.

Generation Procedures. A generation procedure is a precise description of how to derive application work products consistent with the decisions in an Application Model. CCSD automated the generation procedure to provide an Application Engineering environment. The environment consists of a graphical interface and a generation facility, both developed using WordPerfect merge and macro functions. The graphical interface, which consists of a sequence of menus and screens, allows the engineer to create, modify, and browse through an Application Model. The specification language is based directly on the decision model for the domain. The generation facility uses the Application Model to guide the automatic generation of a software requirements document from adaptable requirements components. The adapted documentation is generated by performing a WordPerfect merge of the adaptable requirements components and the engineer's decisions from the Application Model.

Figure 5 lists fragments of the generation procedure (implemented with WordPerfect macros) for generating an SRS. The first merge operation loads previously recorded decisions (from file sysxxx.sf!) made by an application engineer. The second merge operation instantiates the adaptable SRS component (in file srsshell.pf!) with the decisions to produce the actual SRS (in file srs.wpf).

```
...
FORMTYPE:=2
FORMFILE:= "set_syst.pf!"
DATATYPE:=2
DATAFILE:= "sysxxx.sf!"

MergeRun(FORMTYPE;FORMFILE;DATATYPE;DATAFILE;OUTTYPE)
....
FORMTYPE:=2
FORMFILE:= "srsshell.pf!"
DATATYPE:=0
OUTTYPE:=2
OUTDATA:="srs.wpf"

MergeRun(FORMTYPE;FORMFILE;DATATYPE;;OUTTYPE;OUTDATA)
...
```

Figure 5. WordPerfect Macro Fragments for Generating an SRS

3.1.2.3 Application Engineering

Application Engineering is a process for creating and supporting an Application Product that satisfies specified customer needs. The application engineer develops a product by resolving the requirements and engineering decisions as identified in the decision model. The result is an Application Model. The Application Model is then used to drive the generation of the work products from the Adaptable Components. The WordPerfect-based environment developed by CCSD automates this process of Application Modeling and product generation. Figure 6 shows one of the screens from the graphical interface, corresponding to the decision class shown in Table 2. Figure 7 shows a portion of a generated software requirements specification that was produced by tailoring the Adaptable Component shown in Figure 4 using the decision choices shown in Figure 6.

3.1.3 Tooling Improvement Opportunities

CCSD's approach to managing variation and automation, as illustrated in the walkthrough, demonstrated that off-the-shelf tools can be used effectively to provide low-cost support for the

Subsystem Information	
1. Subsystem Identifier:	Mission Control 1
2. Processor:	Intel 80386
3. ROM:	4096K
4. RAM:	4096K
5. 1553B Chip Set:	CT-1612
6. Save Subsystem	

Figure 6. A Screen From the Application Engineering Environment

developers to rapidly specify and generate systems in their domains. CCSD's domains were amenable to CCSD's automation approach; that is, the variations largely consisted of alternate configurations of similar systems allowing for simple substitution of alternate solutions. CCSD's automation approach may be less effective in domains where there are more complex variations.

> 1.2.1.1. Mission Control 1 configuration. The RICC application software and the interprocessor communications software in the Mission Control 1 subsystem will be stored in ROM and executed on an Intel 80386 processor. The Mission Control 1 subsystem contains 4096K of ROM and 4096K of RAM. The RICC application uses the MIL-STD-1553B bus to exchange messages with applications existing in other subsystems in the network. The RICC application communicates over the MIL-STD-1553B bus(es) via the terminal(s) described below.

Figure 7. A Generated Requirements Document Component

The following list describes additional opportunities for automation support, which build on the capabilities established by CCSD:

- ***Variation Representation and Analysis.*** Current computer-aided software engineering (CASE) tools, such as OpenSelect, do not have a built-in capability to represent variation. CCSD was able to work around this problem using annotations and replication. In CCSD's domains, this approach was sufficient; however, as variations become more complex, this approach may become more complicated. To further assist engineers in managing complex variations, tools are needed that support variation analysis (e.g., for completeness and consistency) and generation of different instantiations. Providing this capability would require that tools be able to perform operations on variations as well as provide a notation to represent them.

- ***ADARTS Support.*** Drawing tools, such as GEM Draw, do not understand the semantics of the ADARTS notation and, thus, have no ability to assist a developer in analyzing and verifying their designs. Tool support for ADARTS does exist; however, for CCSD's technique to manage variation, the tool needs to have the ability to decompose a design into multiple levels of abstraction so that variabilities may be isolated.

- ***Tool Integration.*** CCSD used their WordPerfect-based Application Engineering environment to generate SRSs. These specifications incorporate the data flow diagrams and process specifications that CCSD developed with OpenSelect. To accomplish this, however, CCSD must manually export the diagrams from OpenSelect then manually import them into WordPerfect. The result is that CCSD now must ensure that both copies are the same; that is, when any changes are made to a data flow diagram, it must be exported and imported again to ensure integrity. If these tools were integrated so that the diagrams could be shared, then the integrity could be ensured and unnecessary manual procedures could be eliminated.

- ***User Interface Generation.*** CCSD currently uses WordPerfect macros to implement the user interface for Application Modeling and work product generation. The number of macro programs to implement this interface is directly proportional to the number of variations. The use of macros works well when the number of variations is reasonably small, but may become unwieldy when the number of variations grows large. An alternative approach would be to generate the user interface from the Decision Model.

- *Decision Modeling.* As seen in CCSD's approach, the Decision Model impacts many activities and products. Thus, tools that support the creation and use of the decision model could have a significant benefit to domain and application engineers. Possible areas of support include:
  - Maintaining traceability of decisions to variability assumptions
  - Tracking dependencies between decisions
  - Analyzing decisions for completeness, consistency, and redundancy
  - Managing model decomposition and iterative development by multiple teams

3.2 BOEING/NAVY STARS DEMONSTRATION PROJECT

The Advanced Research Projects Agency (ARPA) STARS program was established in 1983 to address pervasive and severe cost, schedule, and quality problems associated with all aspects of software development. The program mission is to demonstrate megaprogramming, a process-driven, domain-specific, reuse-based approach to software engineering that is supported by appropriate tool and environment technology (STARS 1994b). To execute this mission, the STARS program established three teams to jointly develop the concepts and technology necessary to implement megaprogramming. These teams, consisting of major defense systems companies paired with a government service organization, are Boeing/Navy, UNISYS/Army, and Loral/Air Force (formerly IBM/Air Force). Each of these prime teams is chartered to demonstrate STARS technology on real, operational, software-intensive systems.

The Navy/STARS Demonstration Project elected to use the development of the T-34C flight instrument trainer (FIT), sponsored by the Naval Air Warfare Center Training Systems Division (NAWCTSD), to demonstrate the benefits of megaprogramming. The Navy/STARS Demonstration Project also elected to consider the Consortium's Synthesis methodology as an approach to megaprogramming.

In March 1992, the Navy/STARS Demonstration Project initiated a Consortium-assisted pilot project (as part of their Demonstration Project) to evaluate the applicability of a leveraged Synthesis process. This pilot project has continued (and is continuing) through three phases of development on the Navy/STARS Demonstration Project. The Demonstration Project initially tested the Synthesis methodology in a phase called Trial Usage. This phase, which lasted from April 1992 through June 1992, applied Synthesis to a small portion of the Air Vehicle Training System (AVTS) domain: the Environment section dealing with the external environmental conditions of the trainer (e.g., atmosphere). The second application of Synthesis was in a phase called Pilot Usage. This phase lasted from July 1993 through January 1994 and focused on applying the Synthesis methodology to a broader range of trainer functionality: Radio Navigation Aids for T-series aircraft, while also addressing several other goals. These goals included:

- The training of Demonstration Project personnel on the Synthesis process
- The customization of the Synthesis process to the particular needs of the Demonstration Project
- The creation of an initial SEE capability to support Application Engineering

- The education of Navy personnel regarding the management and eventual procurement of systems using a product-line approach for training systems
- Handling geographically dispersed teams for joint development of a domain

The Navy/STARS Demonstration Project is currently in the "Demo" phase of the project's life cycle. In this phase, the project will apply Synthesis to a majority of the functional areas of the T-series trainer domain and will produce a T-34C FIT in 1995 from this domain. This phase will also see improvements, including SEE capabilities, to support the activities of Domain Engineering, and a clearer distinction between Domain Engineering and Application Engineering teams. This phase began in February 1994 and is scheduled for completion in September 1995.

The final capability of this Navy/STARS Demonstration Project will be an automated environment for the specification and automatic generation of software and supporting documentation for T-series trainers. The products will be generated by mechanically selecting, adapting, and composing reusable components based on the decisions expressed in the specification.

The overall reaction (to date) of applying a product-line approach (i.e., Synthesis) to the development of training systems has been positive. Though seemingly complex at times, the megaprogramming approach to developing and maintaining a product line of training systems has begun to yield results. The Application Engineering of systems (for the current state of the domain) is a very rapid, low-cost, and repeatable process, based on the principles of continuous process improvement and systematic reuse.

This section describes the Navy/STARS Demonstration Project's use of tools in the application of the Synthesis methodology.

3.2.1 Tool Usage Summary

Table 3 summarizes Boeing/Navy/STARS's use of tools in support of the Synthesis activities. It indicates the methods and tools used and states how the indicated tools support the corresponding activity. References to "RSP" in the Method column mean that the project performed the activity following only the general guidance provided by the RSP Guidebook (Software Productivity Consortium 1993). References to "ad hoc" mean that the project did not fully apply RSP guidance for an activity.

Table 3. Navy/STARS Tool Usage Summary

Activity	Method	Tool	Tool Usage
Domain Management	RSP PERT	DecPlan Interleaf and IslandWrite	Scheduling and tracking domain development activities. Risk management plans.
Domain Definition	RSP	Interleaf and IslandWrite	Documenting the Domain Synopsis, Glossary, commonality and variability assumptions.
Decision Model	RSP	Interleaf and IslandWrite	Documenting decisions to resolve variabilities in the form of tables and define decision variables.

Table 3, continued

Activity	Method	Tool	Tool Usage
Product Requirements	Ad hoc	Interleaf and IslandWrite	Developing requirements descriptions and diagrams with the descriptions annotated with decision variables to denote variations.
Process Requirements	RSP	Interleaf and IslandWrite	Documenting an automated process for specifying application systems and generating deliverables.
Product Architecture	Structural Modeling	Interleaf and IslandWrite	Documenting and illustrating a DARTS architecture for the product family. Annotated with decision variables to denote variations.
Component Design	Ad hoc	Interleaf and IslandWrite	Documenting specifications for Adaptable Components. Parameterized with the decision variables.
Generation Design	RSP	Interleaf and IslandWrite	Documenting mappings for building systems.
Domain Verification	RSP	DECAda Specialized ADAPT routines KAMEL Denali	Testing Adaptable Components by applying specific decisions to an ADAPT routine to produce Ada-compilable component instances.
Component Implementation	RSP	Interleaf and IslandWrite DEC Editors, DECAda	Documenting the component's implementation. Implementing adaptable Ada software components.
Generation Implementation	RSP	Interleaf and IslandWrite Element Editors and Navigators	Documenting the generation procedure. Populating the SEE with CLIPS rules (part of KAMEL) for performing retrieval and adaptation of components.
Process Support Development	RSP	Interleaf and IslandWrite Element Editors	Documenting the standards, guides, and training for Application Engineering. Populating the SEE with CLIPS rules (part of KAMEL) for performing Application Modeling (i.e., presenting questions and retrieving decision responses), and constructing process descriptions and code for describing and enacting the SEE processes
Project Support	RSP		

Table 3, continued

Activity	Method	Tool	Tool Usage
Application Engineering – Basic Infrastructure	RSP	Amadeus, CDD/Administrator, CDD/Repository, DecPlan, DECWindows, Interleaf, Process Engine, ROAMS, VMS	Specifying applications and automatically generating work products.
Application Engineering – Project Management	RSP	Amadeus, DecPlan, Process Engine	Management of specifying applications and automatically generating work products.
Application Engineering – Application Modeling	RSP	Element Editors, KAMEL, Navigators, Process Engine	Specifying applications.
Application Engineering – Application Production	RSP	Process Engine, ROAMS	Automatically generating work products.
Application Engineering – Delivery and Operations Support	RSP	NA	NA

As the table indicates, the Navy/STARS Demonstration Project relied principally on the Consortium's RSP Guidebook (a.k.a., Synthesis) for process guidance. The Demonstration Project recognized the value of Synthesis as a disciplined engineering process and decided to augment the guidebook with domain-specific extensions (i.e., the next lower level of process guidance).

The Navy/STARS Demonstration Project has primarily focused their automation efforts on the AE process which is embodied in a SEE. Their environment is built upon a rather extensive toolset available on DEC platforms. Through use of DEC tools, integration standards, and custom tools, the demonstration project was able to produce a turnkey environment that allows a team of application engineers to interactively specify decisions distinguishing an application and generate the application work products from Adaptable Components. The Application Engineering environment also integrated many capabilities that exist separately in other environments, namely measurement, project management, and configuration management.

Users of the SEE do not actively need to address the collection of metrics or the management of versions of work products. All of this is handled behind-the-scenes and unobtrusively. Project management and project tasking drive what application engineers work on at any given point in time. Users are assigned tasks that are activated when all necessary preconditions exist and are not allowed to "complete" until all required steps have been performed. Extensive on-line descriptions of the user's tasks and processes help guide the user in performing tasks.

In the upcoming phase of the Navy/STARS Demonstration Project, the SEE will begin to provide some support for Domain Engineering activities. Currently, however, Domain Engineering is still largely a manual process with tool support primarily documenting results. Notable exceptions here

include PERT tools for project management, fundamental navigating/editing tools available through the SEE framework, and some custom tools for aiding component adaptation.

Tool integration is a major focus in the Navy/STARS SEE. The SEE infrastructure, being based upon an integrated DEC toolset, comes with a lot of tool integration already packaged by DEC. This allowed for seamless integration of DEC planning, editing, and storage/retrieval tools. However, additional tooling requirements were necessary to meet the needs of the Navy/STARS Demonstration Team and of automation of parts of Synthesis' Application Engineering process. The Navy/STARS SEE developers integrated non-DEC tools into the SEE based on an extended ATIS (A Tool Integration Standard) type hierarchy for object-oriented information sharing among tools.

3.2.2 Process Artifact and Tool Walkthrough

This section presents selected Synthesis activities and artifacts and discusses how tooling helped the Navy/STARS Demonstration Project to accomplish a Synthesis process.

The following descriptions (Sections 3.2.2.1 through 3.2.2.3) reflect the tooling capability that existed through the Trial Usage and Pilot phases of the Navy/STARS Demonstration Project. However, in the current, and final, phase of the Navy/STARS Demonstration Project, active support for Domain Engineering activities will be incorporated into the SEE. Initially, this will likely take the form of electronic traceability between Synthesis artifacts according to the interrelationships described in the RSP Guidebook (Software Productivity Consortium 1993). Eventually, though, the SEE support for Domain Analysis activities is expected to provide interactive assistance in performing the various activities where the assistance is activity-sensitive and at the appropriate level of abstraction to be useful to the domain engineer.

3.2.2.1 Domain Analysis

Most of the Domain Analysis activities on the Navy/STARS Demonstration Project were accomplished by engineers drafting work products on paper and then capturing the results in some electronic form. The preferred electronic medium was Interleaf on DEC platforms and IslandWrite on Sun platforms. The exception to this was in the area of project management where DecPlan was used for performing project management. PERT charts were available in DecPlan for laying out activity and resource time lines and allocations.

The tabular formatting capabilities in tools such as Interleaf, however, proved to be a good organization mechanism for the tabular information contained in some Synthesis artifacts (e.g., Decision Models and Generation Design).

3.2.2.2 Domain Implementation

The activities of Domain Implementation are split into those implementing the product and those developing the process support.

The primary tools used for accomplishing Product Implementation were conventional editors and compilers. Even though text editors on the various platforms were available for the creation of the Product Implementation, all of the work was eventually migrated to the DEC platforms where it was compiled, debugged, and stored.

The Process Support Development was more nonconventional, consisting of the development of the underlying SEE framework followed by the population of the SEE with the products of Domain Engineering (i.e., Adaptable Components and support for the activities of the Application Engineering process). The primary SEE development (performed by Boeing in Seattle, WA), occurred entirely within the DEC environment (including DEC compilers, editors, and CM).

The primary interface to the SEE used for population of the SEE was through Navigators and Element Editors. Navigators are essentially browsers for viewing repository information in various ways; Element Editors are used to access and/or modify particular objects within the repository. Though this helped to automate some of the process, it was still largely a manual process. The SEE population staff needed to perform activities such as:

- Creating the domain-specific activities and activity descriptions
- Placing the Adaptable Components into the repository
- Writing KAMEL code for retrieving and adapting the components

Even though all of these activities enabled a much more efficient Application Engineering process, the tool support for these Domain Engineering activities was minimal. However, some of the components of the SEE, designed to improve the Application Engineering capability, proved beneficial to the domain engineering staff. One of the custom tools produced by the SEE developers, KAMEL, enabled the creation of screens for obtaining decisions (i.e., an Application Model) from an application engineer. KAMEL code (such as that shown in Figure 8) is Lisp-like, but is able to produce interfaces (such as that shown in Figure 9) that will handle the interaction with the application engineer for retrieving answers to questions (i.e., decisions). Another custom tool performed adaptation of Adaptable Components within the SEE. This adaptation tool was used by the Product Implementation staff for testing Adaptable Components by creating specific instances of Adaptable Components that were then tested conventionally.

3.2.2.3 Application Engineering

The major benefit of the Navy/STARS Demonstration Project's development of the integrated tooling available in the SEE is realized in Application Engineering. The final Application Engineering process support provided by the efforts of Domain Engineering is truly a turnkey process for producing training systems based on systems engineering decisions. This section walks through a development scenario for building a training system artifact (code and documentation) to illustrate how the tooling helps in the specification and generation of end work products.

However, before tracing through the AE development scenario, it is essential to understand that many of the SEE tools are transparent or unobtrusive to an application engineer. The SEE is based upon an infrastructure of commercial off-the-shelf (COTS), public domain, and Boeing/STARS-developed tools integrated to provide the complete Application Engineering capability available to an application engineer. This capability is built on and compliant with several standards (POSIX, Motif, ATIS, and OMG CORBA) which increases the likelihood that the SEE can be expanded and interoperate with other tools. Table 4 illustrates the tools that directly support the Application Engineering development scenario described later in this section.

The STARS SEE is invoked via a menu pulled down from a standard DECWindow session manager. Once the SEE is activated, its Process Engine guides users in performing their work. It uses work

breakdown structures and precedence networks defined through DecPlan to control the precedence of the activities and guide the user through performance of the activities. The Process Engine also collects process metrics for later analysis and process improvement (STARS 1993). Access to the reuse library, also a part of the process, is augmented by a knowledge-based tool, KAMEL, which uses rules about the domain to perform selection and adaptation of system components, as will be described in the Application Engineering development scenario under Application Modeling. Configuration management and version control are provided automatically as each user access objects from the repository. The SEE maintains a separate user work area so that each user will not accidentally corrupt work being done by others. The SEE is a multiuser platform that can support users who are geographically separated.

```
(deffunction make-TACANSelfTest (?spawninst ?value)
    (if (eq ?TACAN_C_answer_value Yes) then
        (bind ?value Copilot)
        (bind ?nvalues (+ ?nvalues 1) )
    )
    (if (eq ?TACAN_P_answer_value Yes) then
        (bind ?value Pilot)
        (bind ?nvalues (+ ?nvalues 1) )
    )

    ...

    (bind ?newsym (sym-cat ?value "TACSlfTestInit"))
    (bind ?newdesc (str-cat ?value "TACSlfTestInit"))

    (bind ?newnam (symbol-to-instance-name ?newsym))
    (bind ?qnewsym (sym-cat ?newsym "Question"))
    (bind ?qnewnam (symbol-to-instance-name ?qnewsym))

    (make-instance ?newnam of MULTI_DECISION
        (Design_Group_ID TACANSelfTest)
        (Question_Name ?qnewsym)
        (One_Line_Desc ?newdesc)
        (InstanceInstantiatedBy ?spawninst)
        (FurtherQFunction make-buttonpress)
        (Multi_Entry nil) )
    (make-instance ?qnewnam of MULTIPLE_CHOICE_QUESTION
        (Decision_Name ?newsym)
        (Question (format nil "%s%s%n%s" "How is " ?value
            "TACAN self test Initiated ?"))
        (One_Line_Question ?newdesc)
        (Text_ChoicesPush_And_Hold Power_On Push_And_Release Other)
        (Choices      Push_And_Hold Power_On Push_And_Release Other)
        (Lines_Selected 0)
        (Decision_Help_Text (format nil "%s%n%s"
            "For initiating self test there are several options. There options are "
            "based on the type of control unit under consideration."))

    ...

    )
```

Figure 8. A KAMEL MAKE Function

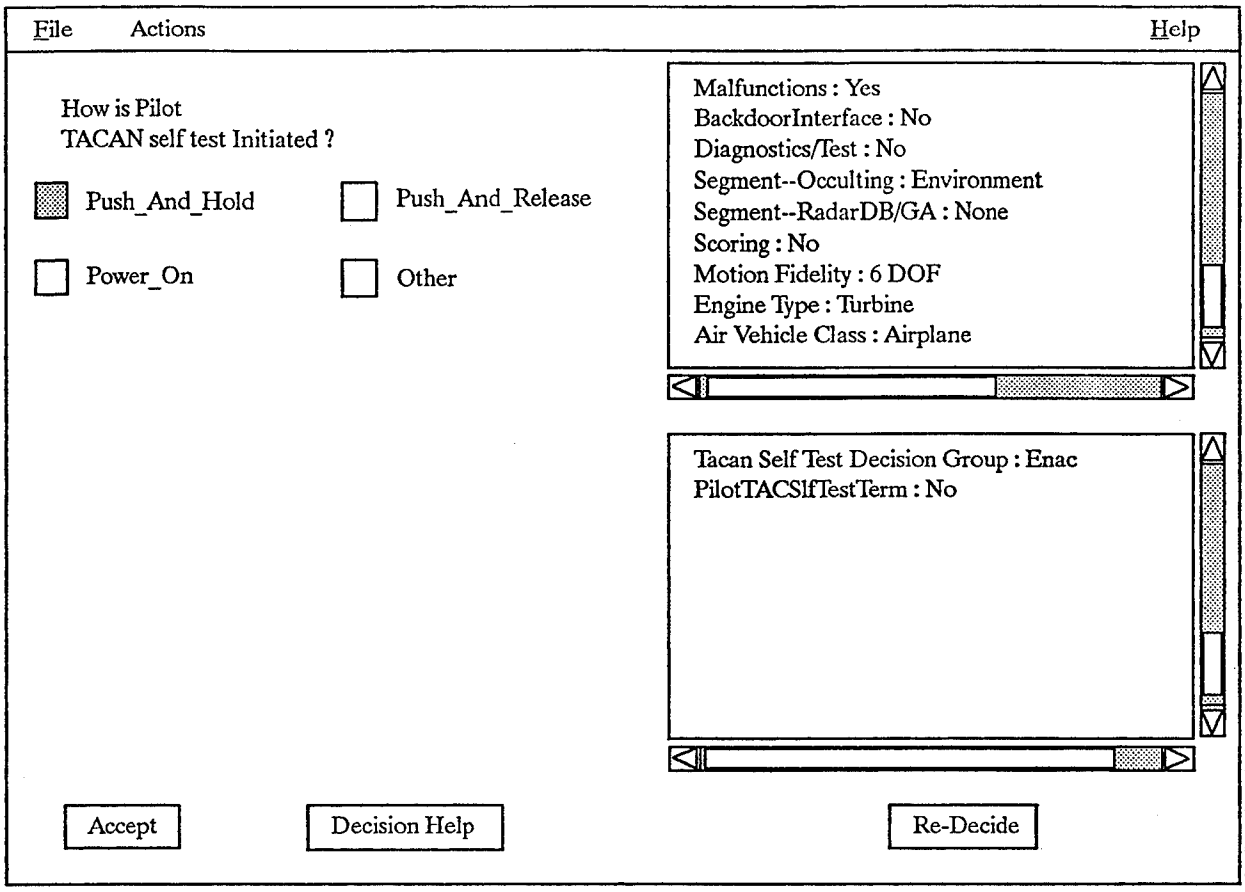


Figure 9. A KAMEL Decide Screen

Table 4. SEE Infrastructure Summary

Tool Provider	Tool Name	Tool Description
Commercial-off-the-Shelf	Amadeus	Measurement
	CDD/Administrator	Repository administrator's tool, navigators, editors
	CDD/Repository	Repository, version, and configuration management
	DecPlan	Management
	DECWindows	Windowing system
	Interleaf	Word processing
	VMS	Operating system
Public Domain	CLIPS	Inference engine used within KAMEL
Boeing/STARS	KAMEL	AE question/answer management
	Process Engine	Process enactment capability providing the integration of an object-oriented repository, a process definition language, a graphical user interface, and a metrics collection capability
	ROAMS	Reuse library mechanism for viewing, accessing, and adapting components within the AE repository

Project and Activity Management. Users (i.e., application engineers) operate within the SEE through SEE-enacted activities. The activities are encoded by domain engineers using the process programming language Agents, Artifacts, and Activities (AAA) that is implemented by the Denali persistent programming language (STARS 1994b). This implementation of AAA has been named AAA+. The code that defines the behavior of the activity is executed on every invocation of the activity. Typical operations include variable initialization, context initialization such as opening a context, and opening a persistent process. Control points trigger off the runtime state of the activity and are utilized to control the interaction between user and the activity. Figures 10 and 11 show how AAA is used to create a user dialog box for retrieving information from the application engineer.

```
PROCEDURE edit_charge_line IS
--
-- Begin local variable declaration
--
wp_Id : STARS_PM_WORK_PACKAGE;
handle : db_scan;
Charge_Line : db_string;
--
-- End of local variable declaration
BEGIN -- edit_charge_line
--
-- Query user for Chargeline
--
Charge_Line := call_stars_Modal_Prompt_Dialog(
    the_title => "Charge-Line Query",
    the_default_text => "---",
    the_msg => "Please input the Charge-Line to be used"
        & "for the " & my_wp.STARS_PM_WPPROJECTNAME
        & "!"
    );

    ...

END edit_charge_line;
```

Figure 10. AAA (Denali) Process Enactment Code

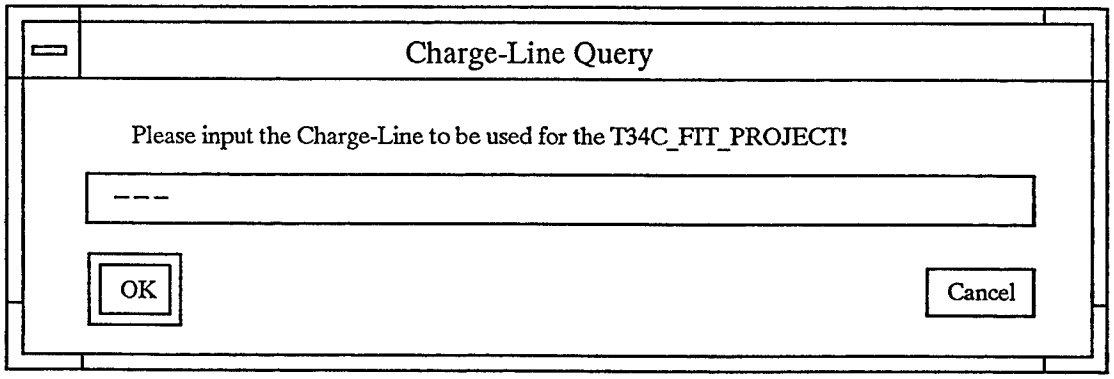


Figure 11. Resulting Charge Line Query Dialog Window

The SEE understands the identification or role of a particular user and presents a list of activities that the user should perform. This individual activity management, enacted by the Process Engine, consists

of activities that contain entry preconditions and required postconditions. The user is not allowed to perform certain activities until that activity's required preconditions are satisfied. Likewise, the user is not allow to indicate an activity is completed until all of the activity's required postconditions are satisfied. Within activities, the user must perform certain actions, known as Required Actions, to complete the activity. Optional Actions are also available to the application engineer, which provide such capabilities as reviewing domain work products (e.g., requirements, architectures), schedules, and repository contents. These Optional Actions are available so that the user can become familiar with the domain or project prior to performing mandatory steps of the activity.

The SEE activity management also contains the notion of "projects." Projects, to complete a particular flight trainer system, are initiated by a project manager who in turn initiates project activities for other engineers on the project. Work breakdown structures and precedence networks, provided through DecPlan, are used to control the activities of the project and guide the user through their particular activities.

Activity-related instructions assist the user in performing a given activity. These instructions provide information such as:

- What prerequisites are necessary for the activity
- What actions are required for a given activity
- What actions are optional for a given activity
- What experience level is necessary to complete the activity

This information is presented to the user through Element Editors and Navigators, provided as part of Digital's CDD/Administrator. The activity-related instructions and user help were based on the structure of the RSP Guidebook (Software Productivity Consortium 1993). The guidebook was used as a framework (i.e., outline) for the processes and descriptions, but was augmented with domain-specific information and instructions to aid an application engineer in the AVTS domain.

As users complete given activities, measurement data is obtained from the user by a measurement tool named Amadeus. Wall-time measurements are taken while users are performing activities, but the user is able to override this system-measured activity time.

As users begin working on an initiated project, separate work areas are maintained by the SEE. All work products retrieved and/or altered by the user are maintained in a user-specific workarea. All work products retrieved from the repository by the user have their version numbers automatically increased to maintain a distinction between the retrieved component and those in the repository.

Application Modeling. Within the SEE, Application Modeling is simply an activity to be assigned and performed by a given application engineer. As such, the application engineer will be presented activity-specific descriptions about what is needed to perform Application Modeling. The user, through the use of Element Editors and Navigators, can perform optional steps, such as reviewing the Domain Architecture, Adaptable Components, or any other portion of the domain in preparation to making the decisions required in Application Modeling.

Once the engineer decides to begin Application Modeling, the underlying tool, KAMEL, performs the interfacing with the user for the decision-making process. KAMEL presents the user with

questions (using an interface with mouse-driven, point-and-click interaction) and discrete answers to select using the mouse or areas for typing in textual responses. Each decision, as it is made, is displayed in a separate pane of the Application Modeling window so that the user can see, review, and even alter previous answers that have been made during this Application Modeling session. The order of the questions is determined by the Inference Engine, and decisions made about one question may determine whether another question is asked.

Once the application engineer has answered all the questions, the responses can be checked/assessed for consistency and completeness. It is only upon a successful check that system work products can be produced based on these responses. This prevents the attempted generation of a system from an incorrect or incomplete Application Model.

Application Production. Once the Application Model is successfully checked by the SEE, the user can initiate the generation of work products based on the Application Model. The generation is performed by a cooperation of the Process Engine and ROAMS. The Process Engine, including the process enactment of Denali, responds to the generation command by supplying retrieval and adaptation directives that need to be performed. The typical adaptations that can be performed by the adaptation directives include parameter substitution, text selection (i.e., word and paragraph), instantiation, and other preprocessing. Figure 12 illustrates some retrieval and adaptation directives generated for building a portion of a System/Segment Specification for the T34C FIT.

```
begin
    ADAPT $Aircraft_Type$ "T-34"
    COPY STARS_RQT_FUNCTIONAL_CAPABILITY INS_BY_NAV_COMM
        nc_INS_BY_NAV_COMM
end

    ...

begin
    ADAPT $AHRS_FOR_CREW_POS$ "Pilot"
    COPY STARS_RQT_FUNCTIONAL_CAPABILITY AHRS_FOR_CREW_POS
        nc1_AHRS_FOR_CREW_POS
end

    ...

begin
    SETPROP MCS_processingName "EXERCISE_DG_AHRS_FOR_Pilot"
    ADAPT $decision_group_name "EXERCISE_DG_AHRS_FOR_Pilot"
    ADAPT $DG_Value_Parent$ "NAVCOMM"
    ADAPT $DG_Definition$ "AHRS_FOR_Pilot"
    COPY STARS_PM_ACTIVITY_PROGRAM EXERCISE_DECISION_GROUP
        AP_AHRS_FOR_Pilot
end

    ...
```

Figure 12. KAMEL-Generated Retrieve/Adapt Directives (Excerpt)

The process enactment code drives whether the retrieval and adaptation directives are automatically performed by the Process Engine or assigned to other user's activities (i.e., as Required Actions). Whether assigned to another engineer or performed automatically after the decision-making activity

is completed, the SEE performs the ROAMS retrievals and adaptations with little work on the part of the application engineer.

Aside from being able to adapt and generate simple text files (e.g., code), the SEE has been able to interact with and produce adapted Interleaf documentation based on the decisions made by the application engineers. The adaptations were made to ASCII versions of the Interleaf documents, and then collated by Interleaf when the files were opened after adaptation. Figures 13 and 14 illustrate

```
<"Section.5">
NAVIGATION_COMMUNICATION

<"para:1">
Navigation_Communication: The simulation of the navigation, communication, and identification systems for
the application aircraft.

    ...

<"Section.6">
NC1_AHRS_FOR_CREW_POS

<"para:1">
The Navigation/Communication segment shall provide the functionality that simulates the AHRS for the Pilot.

<"Section.6">
NC_IFF_BY_NAV_COMM

<"para:1">
The Identification Friend or Foe (IFF) function shall model the operation and functionality of the IFF system
installed on the T-34.

    ...
```

Figure 13. Interleaf ASCII Generated From Retrieve/Adapt Directives (Excerpt)

```
    ...

3.2.1.1.1.2 NAVIGATION_COMMUNICATION

Navigation_Communication: The simulation of the navigation, communication, and identification systems for
the application aircraft.

    ...

3.2.1.1.1.2.4 NC1_AHRS_FOR_CREW_POS

The Navigation/Communication segment shall provide the functionality that simulates the AHRS for the Pilot.

3.2.1.1.1.2.5 NC_IFF_BY_NAV_COMM

The Identification Friend or Foe (IFF) function shall model the operation and functionality of the IFF system
installed on the T-34.

    ...
```

Figure 14. Final Generated Interleaf System/Segment Specification (Excerpt)

the intermediate ASCII Interleaf adapted text and the final Interleaf document, respectively, generated from the directives shown in Figure 12. The actions of this scenario were simple menu selections (i.e., Required Actions) where the Process Engine performed the actual work of the activity.

3.2.3 Opportunities for Improvement

The Navy/STARS Demonstration Project's SEE is an integrated set of COTS, public-domain, and custom-developed tooling that provides a complete turnkey capability for application engineers to construct training systems. The SEE allows application engineers to rapidly specify and generate systems in their domain with project/activity management, measurement, and configuration management being performed seamlessly. The SEE's capability was the result of a major investment in technology support for a product-line development approach to reuse and system building.

Even with all of the SEE's capabilities, additional opportunities exist for automated support. An additional opportunity, which builds on the capabilities already established by the Navy/STARS Demonstration Project's efforts, is to improve support for Domain Engineering. Currently, this support in the SEE is minimal. However, this is due to change within the currently scheduled iterations of domain (and SEE) development. The SEE will begin to help the domain engineer in constructing their work products. At first, this will likely take the form of traceability assistance among Domain Engineering work products to allow for better impact and change analysis. It is desired, however, to improve this assistance to provide a more interactive work-product-building capability where the SEE is prompting the domain engineer for information to help construct the work products for the domain.

3.3 CASE STUDIES ANALYSIS

This section discusses the rationale for the different method and tooling choices that the Rockwell and Boeing/STARS pilot projects have made. Insights resulting from this analysis led to the ideas on tool usage provided in Section 4.

The case studies demonstrate two different scenarios in method and tool selection. Rockwell selected a relatively low investment in employing commercial tools to create a modest, customized environment, whereas Boeing/STARS engaged in a relatively large effort to develop a environment. In each case, these choices were determined by characteristics of the prevailing situation. The key factors were:

- Organizational objectives
- Available investment capital
- Existing tool base

3.3.1 Methods

Rockwell discovered that practicing a Synthesis approach highlighted the value of institutionalizing standardized requirements and design methods. Moreover, having the entire organization adopt standard methods provided immediate value and served as a step in transitioning to Synthesis. Furthermore, the adoption of standard methods opened the potential for cost-effectively employing additional tools.

In contrast, the domain targeted by the Navy/STARS demonstration has no existing, industry-wide standard methods. Even given the obvious advantages to adopting standard methods, having NAWCTSD impose such a standard on its contractors raises a variety of concerns which may be difficult to address, at least in the near term (although the "imposition" of DARTS as the standard architecture within this domain will demand that some of these concerns be addressed). For Boeing to have built in a bias toward specific methods in its first version of the SEE would undoubtedly increase resistance to its acceptance. As it stands, the demonstration will be able to show the benefit of a Synthesis approach used with traditional, ad hoc methods.

3.3.2 Tools

Rockwell's primary objective was initially to validate Synthesis as a viable and effective approach to software development in a domain representative of their business area. As the pilot team's experience and confidence in Synthesis grew, their goals shifted somewhat to increasing emphasis on selling the Synthesis approach to the other members of their division and transitioning the technology to production projects in broader domains. However, the investment level for the project, while not insignificant, was still modest, being provided wholly by internal research and development (IR&D) funds; greater funding would not be available until the approach was sufficiently institutionalized across production projects to be generating a self-sustaining cash flow. Thus, they needed to present the results of their efforts as an Application Engineering environment with an attractive, easy-to-understand, interactive interface (this was a crucial factor in making a positive first impression on the other members of their organization and establishing a compelling case for the value of Synthesis). The only way to achieve this on a modest budget was to utilize the organization's available base of commercial tools and processors. Using tools that were a de facto standard for the organization, or at least familiar to the other staff members (although these tools were sometimes employed in unfamiliar ways), was more attractive and credible than a completely unfamiliar or internally built prototype tool would have been.

The Boeing/STARS program, on the other hand, had a major goal to produce a process-driven SEE. This was seen as a technology necessary to support a product-line approach to software development and achieve its wide spread adoption. The SEE is ultimately intended to support a variety of programs working with diverse contractors in many different domains. Thus, there was no specific tool standard or de facto infrastructure which constrained the SEE, except the general desire for the SEE to eventually be supported by a commercial vendor. This more ambitious objective was pursued with a commensurately larger budget. The budget was invested in building a custom layer of functionality on top of a commercial product base.

This page intentionally left blank.

4. ENHANCED TOOL SUPPORT

This section describes how, based on pilot project and other experience, commercially available tools can be applied to increase the automated support provided for activities of a leveraged Synthesis process. The tools identified are not specifically recommended or evaluated, either among themselves or relative to other commercially available tools that are not listed. Furthermore, the accuracy and completeness of individual tool descriptions have not been validated with their vendors and should not be considered authoritative. Appendix F provides limited additional vendor and platform information on each of the tools listed in this section. An organization should select tools based on their needs and the suitability of available tools to those needs, both in support of leveraged Synthesis and for other uses.

4.1 A TYPICAL TOOL BASE

For context, consider the types of tools that are commonly found in software development organizations. Most organizations have acquired tools to support the following needs or capabilities:

- Software work products (including project plans, requirements, design, code, testing and installation materials, and user documentation) stored in computer files
- Reusable assets (fragments of potential work products of future projects) stored as computer files in a hierarchical directory structure
- Document processing/publication (such as Microsoft Word, WordPerfect, Interleaf, FrameMaker)
- Project management (such as DECPlan, MacProject, Microsoft Project for planning)
- Programming (editors, compilers, linkers, builders, such as vi, cc, ld, make in UNIX environments)

Particular organizations may also have additional types of tools such as:

- CASE tools and frameworks (such as **team***work*, Software Through Pictures, SoftBench, ObjectMaker, OMTool, Objectory, PowerBuilder, Information Engineering Facility, Cohesion, SNAP, Statemate, ObjecTime, MatrixX)
- Database tools (such as Oracle, DB2, Microsoft Access, FileMaker Pro, Paradox)
- User interface builders (such as Galaxy, XVT, Open Interface, NextStep)
- Testing and simulation tools

Because the ability to profitably use tools that are already in hand would ease the acceptance of a leveraged Synthesis process, this survey of tool ideas gave added weight to identifying how these tools could support a leveraged Synthesis process.

4.2 TOOLS FOR DOMAIN MANAGEMENT

Domain Management is an activity for planning, directing, and controlling business-area resources to achieve business objectives. In this respect, Domain Management is very similar to conventional program management. Thus, many of the existing techniques and tools for project management may also be applied to Domain Management. What differentiates Domain Management from project management is the added complexity that comes from coordinating domain development activities with multiple application engineering projects. This section identifies project management tools that can be used for domain planning, process management, and configuration management in support of the planning, directing, and controlling activities of Domain Management.

4.2.1 Tools for Domain Planning

The object of domain planning is to organize resources (time, budget, and people) to achieve near- and long-term domain objectives. The key to successful domain planning is to effectively apply the organization's resources to the family of systems rather than to independently plan each system. Existing planning tools can be used in this regard. Many planning tools have the capability to support multi-project planning so that the plans can be managed collectively as a family. Other planning tool capabilities include support for:

- Scheduling domain activities
- Identifying dependencies between domain activities
- Allocating budget and personnel resources to domain activities
- Measuring and recording progress
- Reporting domain status

Some project planning tools that can be applied to domain planning include:

- CA-SuperProject
- DECPlan
- MacProject Pro
- Microsoft Project
- TimeLine

STSC (1993) provides a comprehensive survey of project management tools.

4.2.2 Tools for Process Management

The object of process management is to ensure predictable cost, schedule, and quality through the definition, execution, and analysis of the engineering process. In the past, process management tools

principally provided capabilities to track reported defects (e.g., QualTEAM). More recently, tools are being developed to directly support the definition, execution, and analysis of a process. The Boeing/STARS Process Engine is an example of such a tool. Typical capabilities for process management tools include support for:

- Specifying a process model
- Notifying engineers of work to be performed and of work completed
- Invoking software tools
- Collecting metrics
- Enforcing product and process standards

Some process management tools include:

- Amadeus
- *PROCESS* WEAVER
- SynerVision
- Virtual Software Factory

STSC (1994) and Eaton (1994b) provide further information on these and other process management tools.

4.2.3 Tools for Configuration Management

The object of configuration management is to ensure the integrity of the set of artifacts that make up a system or family of systems. Configuration management in a domain has the same requirements as configuration management for a project. However, there is an additional complexity in domain configuration management because a reusable asset may be included in multiple configurations. Typical capabilities for configuration management tools include support for:

- Identifying and versioning configuration items
- Baselining configuration items
- Controlling modifications to configuration items

Examples of tools that support configuration management are:

- All Change
- CaseWare/CM
- ClearCase
- CMS and MMS

- RCE
- CCC/Manager
- PVCS

Eaton (1994a) provides further information on these and other configuration management tools.

4.3 TOOLS FOR DOMAIN DEFINITION

Domain Definition is largely an informal, information-gathering, and direction-setting activity. Tooling has utility for recording and interrelating the Domain Synopsis, Domain Glossary, and Domain Assumptions. In addition, tooling can support the organized warehousing and reverse engineering of Legacy Products.

Tools for the purpose of recording and interrelating information need to support extensive free-form and tabular text, graphics, and linkages between related items (i.e., hypermedia links). Capabilities supporting linkages between related items are only beginning to emerge. A tool such as Mosaic exemplifies such a capability. A minimal capability (without linkages) is provided in word/document processor, spreadsheet, and drawing tools (possibly in combination) such as:

- FrameMaker
- Interleaf
- Microsoft Word
- WordPerfect
- Microsoft Excel
- Lotus 1-2-3

Legacy Products can be organized into a taxonomy and stored in a hierarchical directory structure of a conventional file system or they can be managed with a configuration management tool or a reuse repository tool. Repository tools provide more powerful capabilities for locating fragments that might be of use in constructing Domain Engineering work products. Examples of repository tools are:

- Reuse Management System
- InQuisiX

Tools for reverse engineering can assist in analyzing Legacy Products or retrieving fragments in a form closer to that needed by domain engineers. Examples of such tools are:

- Refine
- **Team***work* C/Rev
- Bachman/Analyst Capture

4.4 TOOLS FOR DOMAIN ANALYSIS

Domain Analysis is an activity for informal or semiformal specification and design of a product family and associated production process. Relevant tools support specifying Decision Models, Product Requirements and design, and Process Requirements.

4.4.1 Tools for Decision Modeling

The Decision Model is essentially a database of the decisions that domain engineers are deferring to be made by application engineers. As such, it is a special-purpose database that represents the schema for the Application Model which is essentially a database of the decisions corresponding to a particular Application Product. Any database tool can be used to represent a Decision Model. Examples are:

- DB2
- DEC Rdb
- FileMaker Pro
- 4th Dimension
- GemStone
- Microsoft Access
- ObjectStore
- Ontos DB
- Oracle
- Paradox
- Sybase
- Versant ODBMS

4.4.2 Tools for Product Requirements and Design

The selection of a tool for specifying the requirements and design for a family depends on the software methods an organization has adopted. Commonly supported methods include instances of structured, object-oriented, and information engineering methods. An additional concern in selecting among these tools is whether there is any support for representing variation among a set of systems or versions; in general, however, tools of this sort are equally weak in this respect. Some tools for this purpose are:

- **Team***work*
- Software Through Pictures

- RDD-100
- Information Engineering Facility
- PowerBuilder
- OMTool
- Objectory
- ObjectMaker

4.4.3 Tools for Process Requirements

Tools that support specifying Process Requirements are available but in smaller numbers than other types of tools. Most tools support either IDEF-0 or a proprietary notation for depicting a process. Such tools include:

- RDD-100
- Design/IDEF
- BPwin

4.5 TOOLS FOR DOMAIN IMPLEMENTATION

Domain Implementation is an activity for implementing reusable components from which application engineers can compose tailored systems and appropriate automated support and documentation for the prescribed Application Engineering process.

4.5.1 Tools for Product Implementation

Tools that support Product Implementation are the most familiar types for software developers. These are tools that support creating code, including user interfaces, test support, and documentation.

4.5.1.1 Documentation Tools

Document processors provide comprehensive capabilities for constructing documentation. Most processors provide capabilities for adaptable documentation in the guise of 'form letter' mechanisms. In some cases, TRF2 constructs can be layered on top of a document for more flexible adaptability. Such tools include:

- FrameMaker
- Interleaf
- Microsoft Word
- WordPerfect

4.5.1.2 Code Implementation Tools

There are many tools that support code implementation. Most provide not only the traditional capabilities for editing, compiling, and linking but also integrate capabilities testing, documenting, and configuration managing of implementations. Such tools include:

- Microsoft Visual C++
- Symantec C++
- Borland C++
- SunPro Workshop for C++
- Cohesion
- Rational Rose (C++, Ada)
- Eiffel
- VisualWorks (Smalltalk)
- C++ SoftBench
- ART
- Smalltalk/V

None of these explicitly supports creating and using Adaptable Components, but all provide limited mechanisms that can be used for that purpose (such as templates and macros in C++, generics in Ada, and subclassing/inheritance in any object-oriented language). Tools such as TRF2 and MetaTool, that are specifically oriented to metaprogramming, but lack full commercial support, can be used with these tools if greater flexibility is needed but their use would require additional effort.

4.5.1.3 User Interface Builder Tools

In addition to tools for direct construction of code, there are tools that support visual construction of application user interfaces (that is, the user interfaces of application systems). These include tools such as:

- Galaxy
- NextStep
- OpenInterface
- XVT
- Visual Basic
- Symantec

4.5.1.4 Testing Tools

Most of the code implementation tools include mechanisms for integrating and testing code components. In addition, there are independent tools that support testing. Within Product Implementation, these tools are used to create test scenarios and data, appropriate to a tool, that can be used by projects to test generated applications. Such tools include:

- Logiscope
- McCabe Instrumentation Tool
- Playback
- Purify

4.5.2 Tools for Process Support Development

A primary concern of Process Support Development is to create a framework of tools that application projects can use effectively to perform a standardized Application Engineering process. Tools are needed primarily to support representing Application Models and interactively specifying and viewing those models. This section does not identify other tools needed, particularly ones for evaluating an Application Model, except in that there are a few tools which provide comprehensive Application Engineering capabilities within particular types of domains.

4.5.2.1 Tools for Representing an Application Model

An Application Model is essentially a database of Application Engineering decisions. The Decision Model defines the schema of that database. Numerous commercial database management systems (DBMS) can be used to implement an Application Model. An associated capability, for application engineers to create an Application Model, is often bundled with a DBMS or aided by DBMS facilities for integrating with separately available user-interface tools.

Desirable tool-supported features specific to support of Application Modeling are:

- Representational adequacy (decision classes and all types of decisions can be represented reasonably)
- Multiple concurrent users (a large Application Model can be created by a team of application engineers)
- Schema evolution (modifications to a Decision Model do not invalidate existing Application Models)
- Data versioning (multiple, alternative Application Models or parts thereof can exist simultaneously for direct comparisons of meeting customer needs)
- Application Modeling interface (decisions can be presented to application engineers for resolution and subsequent modification within a reasonable interactive process)

- Dependency maintenance (changes to a decision may invalidate/nullify other decisions; it should be possible for domain engineers to program such side-effects so that they are implicit to application engineers)

- Application Production interface (decisions drive Application Production; it should be possible to interactively query decisions or access decisions from automated Generation Procedures)

- Configuration management (it should be possible to associate an Application Model or version with the corresponding generated product and any hand-modified variants)

There are three classes of systems that can be used to implement an Application Model: file-based, relational, and object-oriented systems. File-based and relational systems are similar in that aggregate (that is, multivalued) decisions must be represented as distinct files/relations, and representation of nested and related decision classes requires matching on replicated, uniquely valued key fields. Object-oriented systems explicitly support aggregate fields and direct referencing of any object (instance of a decision class) via a unique identity, either directly or by navigation through object references.

To implement an Application Model in a file-based or relational system, each decision class is represented as a file or relational table. Each decision is a field of a file record or of a tuple in a relational table. Nested or related decision classes are linked by matching field values. Representable decision values are generally limited to a predefined set of built-in data types. Examples of file-based and relational systems are:

- DB2

- DEC Rdb

- FileMaker Pro

- 4th Dimension

- Microsoft Access

- Oracle

- Paradox

- Sybase

To implement an Application Model in an object-oriented system, each decision class is represented as an object class. Each decision is an attribute of objects in the class. Nested or related decision classes are linked by object identity. Decision classes can also be specialized into subclasses to represent conditional decisions. Representable decision values are usually extensible, from built-in data types to user-defined types. Examples of object-oriented systems are:

- GemStone

- ObjectStore

- Ontos DB
- Versant ODBMS

4.5.2.2 Tools for Providing a User Interface for Application Modeling

While the ability to represent an Application Model is fundamental to an Application Engineering capability, application engineers' view of it depends on the tool that enables entry and viewing of decisions. This tool may be an adjunct to a tool having integral data storage, or it may be independent with a programmatic interface to a database tool. Tools with a bundled capability for user input of decisions include:

- FileMaker Pro
- WordPerfect
- Word

Independent user interface tools include:

- Galaxy
- NextStep
- OpenInterface
- XVT
- Visual Basic
- Ontos Studio

4.5.2.3 Domain-Specific Tools for Application Engineering

As an alternative to putting together an Application Engineering environment out of disparate tools, there are available tools that provide relatively comprehensive capabilities. For domains that are adequately supported by one of these tools, the effort of providing process support can be greatly reduced. Other key issues for these tools are extensibility and tailorability to fit an organization's particular needs, the completeness and quality of producible products, and whether adequate competitive advantage is achievable if the goal is to build commercially competitive applications. Examples of such tools include:

- Matrix-X
- Statemate
- ObjecTime
- SNAP

4.6 TOOLS FOR PROJECT SUPPORT

Beyond the tools provided by Domain Implementation, which Project Support either delivers to application projects or uses to accomplish delivery, there is a need to capture and track problems encountered by the projects in using those tools. Tools that can be used for problem tracking include:

- CaseWare/PT
- CCC/Pro
- ClearQuality
- Lotus Notes

This page intentionally left blank.

5. CONCLUSIONS

Based on the Rockwell and Boeing pilot projects and other experience, the following insights are generally applicable to organizations beginning to adopt a leveraged Synthesis process:

- Familiar, commonly available tools can be used effectively in support of a leveraged Synthesis process. However, comprehensive support currently requires substantial investment in additional automation and tool integration.

- Initial tooling efforts should focus on the needs of Application Engineering. Improving the efficiency and effectiveness of product development is, after all, the principle motivation for instituting a reuse-driven process. Attention to tooling for Domain Engineering comes second, focusing on how to reduce the effort required to create and evolve a domain and Application Engineering Process Support.

- The Application Engineering Activity in a leveraged Synthesis process is substantially simpler than conventional software development. Consequently, appropriate tooling requires relatively modest effort. However, Domain Engineering shares the obstacles to effective automation that are typical of conventional development and introduces additional complexity.

There are several corresponding opportunities open to vendors for improving the support that commercial tools provide for a leveraged Synthesis process:

- Add mechanisms for adaptability, like those found in document processors, Ada generics, or C++ templates, to other tools. These mechanisms support both the creation of adaptable work products and the processing of those into tailored instances. The need is particularly great in tools that support the formulation of requirements and design specifications.

- Extend implementation environments to support domain engineers in creating and verifying Adaptable Components.

- Create project management tools that specifically support an iterative process (i.e., a process in which activities are nonterminating and schedules accommodate explicit cycles through multiple instances of each activity). A milestone in an iterative process is met by the baselining of a work product version, rather than the 'completion' of a work product.

- Provide tailorable tools that domain engineers can specialize to have domain-specific nomenclature, information presentations, operations, and explanation. Alternatively, provide specialized tool components that domain engineers can combine and extend to create a domain-specific tool.

This page intentionally left blank.

APPENDIX A. BOEING/STARS SEE EVALUATION

This appendix describes the conclusions of an initial exploration of the demonstration (Demo) version of the Boeing SEE (STARS 1993). This exploration was initiated to determine the level of support provided by the SEE for a leveraged Synthesis process (Software Productivity Consortium 1993). This note describes the nature and limits of support that the SEE provides relative to the work products and activities of leveraged Synthesis. Section 3 of this report provides a more comprehensive analysis of how the process was performed, as a hybrid manual/automated task, within the SEE framework and identifies which tools supported each of the activities of domain and Application Engineering.

Overall, the Demo SEE provides capabilities that are representative of a well-conceived Application Engineering Environment (AEE). It supports the essential elements of an Application Engineering process in enabling the domain-specific creation of an Application Model and the generation of work products from reusable components guided by the Application Model. Although the Demo SEE supports construction of only limited variations of a single work product family, it does so in a Synthesis-compatible fashion. Consistent with the project's plans and status, support for Domain Engineering is fragmented and at a relatively low level of detail.

The SEE's major shortcoming, which is attributable both to the level of maturity of the SEE and to its having originated as a domain-independent SEE, is that too many aspects of a conventional SEE capability are evident to a user. For example, generic operations related to project and process management, specific to underlying tools, or used for accessing underlying domain information predominate over domain-specific ('process') operations. An advantage of a domain-specific SEE (i.e., an AEE) is that its capabilities are tailored and streamlined to match the user's tasking; the Demo SEE has only begun to achieve this vision.

The SEE's basic mechanisms appear to be sufficient to create an AEE better tailored to support an AE process for the NAWCTSD AVTS domain. More information would be required to adequately judge whether shortcomings in the SEE viewed as an AEE are characteristic of the SEE's capabilities (and therefore difficult to change) or are only an artifact of the particular instantiation of an imperfect process model (and therefore relatively easy to change). The fact that this version of the SEE was created for demonstration purposes rather than engineering usage accounts for some of the following comments.

The assessment analyzed and evaluated three factors:

- Ability of the SEE to portray an AEE:
  - The Demo SEE has representative capabilities of a well-conceived AEE. It is sufficient to show the potential of the underlying mechanisms to support domain-specific Application Engineering.

- The Demo SEE, as configured, is a small fragment of a complete AVTS AEE, consisting of only a single activity thread of project management, Application Modeling, and Application Production, corresponding to a single work product (the SSS) varying with respect to a single accessible decision group. This is an insufficient basis to assess whether the SEE will scale up to full use.

- Validity of the Application Engineering process supported by the Demo SEE:

  - The Demo SEE has certain activities for demonstration purposes only. The most obvious example is the series of three steps that the user (in the role of a technical editor) must go through to generate a work product instance (an SSS) after an Application Model has been completed; in principle, only a single unified step would be necessary. The SEE, configured as an AEE for project use at NAWCTSD, would ideally contain a single unified product generation step. Alternatively, for flexibility but less transparency, it might allow separate generation of individual work products or editing of a generated work product (the latter, however, circumvents Domain Engineering and, therefore, is discouraged as a standard procedure).

  - The Application Engineering process supported by the Demo SEE is excessively serialized and noniterative. An Application Engineering process should encourage experimentation and refinement through iteration. There should be a sense of ongoing, concurrent, communicating activities. Specifically, an AEE should allow redeciding of any decisions, going across all decision groups, at any time. It is unrealistic to expect decisions to be resolved completely or correctly in a single pass. The consequence of modifying decisions is the regeneration of affected work products which are then (re)reviewed/verified and validated. Iteration continues until a satisfactory and complete product has been produced.

  - Similarly, the Application Engineering process supported by the Demo SEE does not accommodate the explicit entry of alternative resolutions of decisions that would specify alternative solutions to a customer's requirements. It should be possible to specify such alternatives and compare the resulting products to determine which best addresses the customer's problem. Decisions often interact in complex ways in the generation of a product, with implications that an application engineer might not foresee. Allowing for the creation of only one solution at a time prevents experimentation and direct comparison of alternatives as a means to finding a best solution.

  - The utility of user roles as a means to structuring and partitioning the activities of the supported Application Engineering process is unclear from use of the Demo SEE. Presumably, a large effort tasked to create a large, multifaceted Application Model resulting in many work products would benefit from an AEE in which there was clear delineation of roles. However, in using the Demo SEE, it seems to force too much delineation. Also, roles and user identities should be distinct concepts to allow multiple users to fulfill each role and each user to fulfill multiple roles.

  - The Application Engineering process is overwhelmed by management infrastructure in the Demo SEE. Substantially more effort is required for project management and time accounting than for actual product engineering. It seems unlikely from use of the

Demo SEE that this infrastructure will be less obtrusive when managing a large project.

- Visibility and effects of conventional SEE underpinnings:

  - The Demo SEE is encumbered by a conventional software engineering orientation. Many of the activities presented to the SEE user in window header menus and buttons are generic and unrelated to the needs of domain-specific Application Engineering. Most of these capabilities represent elements of the SEE infrastructure that are essential at some level, however, few of them are of any direct use in domain-specific Application Engineering. As a result, the initial impression is one of complexity and detail, which obscures the essential Application Engineering process. The Demo SEE takes on the feel of an AEE only after the user descends into the process-specific capabilities of the SEE. Elimination of nonessential generic activities would make the Demo SEE both simpler and more effective as an AEE.

  - Underlying mechanisms are sometimes overly visible to the application engineer. For example, activity and field identifiers are sometimes generic or in process engineering terms (e.g., create/modify 'elements') when domain-specific terminology within the particular context would be more effective. Similarly, window-specific help is often simply a description of the underlying tool when the real issue is the role of the referenced activity in the Application Engineering process. Direct use of generic tools, such as DECPlan in the project management activity, is obtrusive in a domain-specific AEE. Instead, a project plan should be abstracted for the domain and parameterized with decisions, just like other work products; project planning should then consist of making project-specific decisions from which a detailed plan is generated.

  - At the top level, the Demo SEE provides access to underlying domain artifacts. Although accessing those artifacts is often useful to gain deeper understanding of a domain, it is not generally part of product development proper to do this. These information-only actions should be pushed into the lower levels of the Application Engineering process, as help or review aids.

Some additional, minor observations were:

- The process for exercising the NAVCOMM decision group includes a 'practice decide' and a 'formal decide.' Such a distinction seems artificial because refinement of decisions will generally require repeated iteration within and among decision groups anyway.

- Access to underlying requirements and design from each decision group is not a general capability. It is not typical for requirements and design to partition neatly along the lines of the decision groups.

- One of the generic operations is to view the 'Application Model' domain asset. This actually presents the domain's Decision Model, suggesting some confusion in the use of these terms.

This page intentionally left blank.

APPENDIX B. USING FILEMAKER PRO TO REPRESENT DECISION AND APPLICATION MODELS

Claris Corporation's FileMaker Pro database product is an inexpensive, easy to use tool that runs on IBM PC-compatible and Macintosh computers. As such, it is readily available to the majority of Synthesis practitioners. Therefore, it is a candidate tool for a project wishing to use automated tools during Synthesis but forced by budget or other concerns to reject more specialized, expensive CASE tools.

This appendix discusses how domain engineers and application engineers can use FileMaker Pro during Synthesis activities. The focus is on creating and using the decision and Application Models. The discussion is based on comparison with other equally inexpensive tools. Two baselines are used for the comparison. The first is no automation: a project where the decision and Application Models would be captured on paper. The second is a networked computer environment with only rudimentary tools that can capture information but cannot structure it. A text editor would be an example of such a tool.

Table 5 summarizes Synthesis activities in which FileMaker Pro is useful and the ways in which it may be used.

Table 5. Summary of How FileMaker Pro Supports Synthesis

Synthesis Activity	Uses for FileMaker Pro
Decision Model	Domain engineers may use FileMaker Pro to represent the Decision Model. Using it can help ensure its consistency and completeness, and—in group settings—can guard against people entering redundant decisions.
Product Requirements	Domain engineers may use FileMaker Pro as a mechanism to answer queries about the content of the Decision Model and to specify the user interface for Application Engineering tools.
Process Support Development	Domain engineers may use FileMaker Pro to automate portions of Application Modeling.
Application Engineering	Application engineers may use FileMaker Pro to help them create and maintain Application Models.

B.1 OPERATIONS PROVIDED BY FILEMAKER PRO

To facilitate comparing FileMaker Pro to the two baselines, it is necessary to discuss the capabilities FileMaker Pro offers.

FileMaker Pro is a simple database tool. Its user need not be a proficient programmer; its paradigm requires neither algorithm development nor language mastery. It provides a simple, file-based data

model. A database is a set of files, each of which consists of a set of records. Each record consists of a set of typed fields.

In essence, FileMaker Pro provides the following operations:

- *Centralized Data Access (Storage and Retrieval).* In a networked computer environment, FileMaker Pro files can provide a common location for data. (Any tool that lets users enter and retrieve data provides this capability, which is really a feature of the network. However, it is an important, fundamental advantage over paper-and-pencil representations.)

- *Access Control.* A file's creator may assign certain rights to it. Examples of such rights include control over who may read the file, modify the file, or modify the file's structure. (This is somewhat more powerful than the access facilities a network might provide. A network might control read or write access, but it has no control over the structure of a file. Thus, FileMaker Pro offers some advantages over a text editor or other tool for representing unstructured information.)

- *User-Interface Creation.* FileMaker Pro provides a set of icons and drawing facilities consistent with its field-oriented paradigm. Fields may be arranged in any location and may be decorated with graphics.

- *Queries.* FileMaker Pro provides a browsing and query mechanism based on a set of built-in operations. (However, its query facilities are limited compared to the typical database management system. In particular, there is no join operation.)

- *Operation Invocation.* Users may invoke FileMaker Pro's built-in operations, either directly (via menus or keys) or indirectly (via buttons provided by a file's creator). Also, one built-in operation invokes other programs. This lets FileMaker Pro users interact with their environment and, in effect, means that any operation is possible on data entered through FileMaker Pro.

- *Structured Data Collection and Storage.* FileMaker Pro lets a user enter data in a file according to the file's predetermined structure. FileMaker Pro automatically stores data that a user enters.

B.2 USING FILEMAKER PRO DURING THE DECISION MODEL ACTIVITY

The following list describes the FileMaker Pro operations that are useful during the Decision Model Activity:

- *Centralized Data Access.* When a group of domain engineers uses FileMaker Pro during Decision Modeling, everyone allowed to read the files representing the Decision Model has immediate access to the latest version of the Decision Model.

- *Access Control.* The domain engineers will probably want to assign subgroups with responsibilities for different portions of the Decision Model. FileMaker Pro can assign the ability to modify a file to a subgroup. By partitioning decision groups across files, the group capability can ensure that only those people within a group can create or modify files.

- *Queries.* Domain engineers can use FileMaker Pro to determine the state of the Decision Model: what decisions have been created, their type, and so on.

B.3 USING FILEMAKER PRO DURING THE PROCESS REQUIREMENTS ACTIVITY

The following list describes the FileMaker Pro operations that are useful during the Process Requirements Activity:

- *Centralized Data Access.* The files created during the Decision Model Activity give domain engineers a standard place to look when they have questions regarding the Decision Model.
- *Access Control.* Domain engineers can be prevented from accidentally modifying the Decision Model.
- *Queries.* Domain engineers may use FileMaker Pro to answer specific questions about the Decision Model, such as the range of permissible answers for a decision specification or the set of decisions that make up a decision group.
- *User-Interface Creation.* If domain engineers determine that application engineers should see the Application Model as a set of forms, FileMaker Pro provides a means to describe the format and content of those forms pictorially.

B.4 USING FILEMAKER PRO DURING THE PROCESS SUPPORT DEVELOPMENT ACTIVITY

The following list describes the FileMaker Pro operations that are useful during the Process Support Development Activity:

- *Centralized Data Access.* The files created during the Decision Model Activity give domain engineers a standard place to look when they have questions regarding the Decision Model.
- *Access Control.* Domain engineers can be prevented from accidentally modifying the Decision Model.
- *Queries.* Domain engineers may use FileMaker Pro to answer specific questions about the Decision Model.
- *User-Interface Creation.* If application engineers enter the Application Model as a set of forms, domain engineers may use FileMaker Pro to specify those forms.
- *Operation Invocation.* If domain engineers decide to use FileMaker Pro as the front end for the Application Engineering environment, they can create an interface that will allow the application engineer to invoke validation and generation tools.

B.5 USING FILEMAKER PRO DURING THE APPLICATION ENGINEERING ACTIVITY

The following list describes the FileMaker Pro operations that are useful during the Application Engineering Activity:

- ***Data Collection and Storage.*** Application engineers can enter decision specifications as FileMaker Pro field values.
- ***Queries.*** Application engineers can examine existing Application Models.
- ***Operation Invocation.*** Application engineers can use FileMaker Pro to invoke other tools that validate, or generate systems from, Application Models.

APPENDIX C. CREATING ADAPTABLE DOCUMENTATION

This appendix discusses how domain engineers can use a document processor to create adaptable documentation that can be instantiated into tailored documentation by application engineers. The capabilities of Interleaf 5 (Sun), Microsoft Word 5.1 (Macintosh) and 2.0 (Windows), WordPerfect 3.0 (Macintosh), and FrameMaker (Macintosh) are described. It is feasible to use the mechanisms in any of these products to create adaptable documentation. In addition, if a tool can store and retrieve documents in ASCII format, domain engineers can layer adaptability onto a document using the Consortium's TRF2 metaprogramming notation (Software Productivity Consortium 1991).

Sections C.1 through C.4 describe the mechanisms for adaptability supplied by each of these document processors and by TRF2. The motivation for these mechanisms is the need to create a set of documents that are similar and yet tailored to the needs of particular audiences. A simple example of this is form letters.

C.1 INTERLEAF MECHANISMS

Interleaf has two related mechanisms, effectivity control and attributes, specifically designed to support adaptable documents. In Interleaf, attributes are used to characterize the dimensions of variation for a document and effectivity control is used to manage how those attributes determine document tailoring. Each attribute has a user-defined (string or enumerated) value which can either be substituted as document content or be referenced within control expressions to control the inclusion and exclusion of particular document elements (components, inline components, frames, named graphics objects, and tables — substitution is table-wide but rows and columns and individual cell values can be selectively included or excluded). In addition, effectivity control can be applied to linked external documents to define adaptable compound documents.

C.2 MICROSOFT WORD MECHANISM

Microsoft Word provides a "print merge" mechanism that can be used to create adaptable documents and tailored instances. This mechanism, conceived to support tailoring as for simple form letters, includes a macro capability for creating a pattern document consisting of text placeholders, conditional text, and inclusion of other files, in addition to normal document content. Document tailoring entails substituting associated stored (data-document) or user-input decisions into the placeholders and conditionals of a pattern document and macro-evaluating the result to create a tailored document.

C.3 WORDPERFECT MECHANISMS

WordPerfect provides macro and document merge mechanisms that can be used to create adaptable documents. It has extensive macro operations that enable the creation of text placeholders,

conditional and iterated text, and inclusion of other files. Document tailoring occurs via the document merge mechanism, substituting user-input decisions into placeholders, conditionals, and repeating text, and macro-evaluating the result to create a tailored document.

C.4 FRAMEMAKER MECHANISM

FrameMaker provides two mechanisms, conditional text and variables, which support adaptable documents. Conditional text is a text selection that has been associated with (one or more) user-defined named tags that a user can flag as 'in' or 'out' of a document version. Tables, rows, cells, and text-anchored frames can also be associated with these tags. Tags have associated format override options with which users can control the appearance (from a selection of style and color choices) of tagged text. Variables are symbols that have associated text which is substituted into all occurrences of the variable within document text. Text associated with a variable can include intermixed format control selectable from the choices in the document's character format catalog.

C.5 LAYERING ADAPTABILITY ONTO AN INTERLEAF DOCUMENT WITH TRF2

TRF2 is a notation for describing the adaptability of a work product and an associated tool for generating tailored instances of the work product. Within an Interleaf document, users can embed TRF2 constructs, both at the component level and inline, to define how deferred decisions determine variations in document content. TRF2 constructs support text substitution, conditional text, repeating text (containing nested substitutions, conditionals, and repetitions), included subdocuments (from other files), and definition of parameterized 'subdocuments' that can be expanded as tailored instances wherever needed in the containing or including documents. Each of the user-input decisions to a TRF2-instrumented document can be specified to be simple text (retaining any associated formatting properties), a symbol, a structured set of subdecisions, or a repeating set of subdecisions; each can be designated as required or optional (left undefined).

By using a specially defined Interleaf "trf2" component, a user can distinguish instances of these constructs from other, normal components and color-code them to stand out visually from normal document content. After an adaptable document has been created along with a specification of appropriate instantiation decisions (and saved in Interleaf ASCII format to enable filtering and text-based processing by TRF2), the TRF2 tool is applied to instantiate the document, creating a tailored version of the document that can be viewed and printed with Interleaf. This technique can also be applied to FrameMaker documents (after being saved in file interchange [MIF] format for TRF2 processing).

C.6 COMPARISON OF APPROACHES

Common features of all five approaches to adaptable documentation are:

- Any range of text, complete tables, and complete graphical frames can be selectively included or excluded in tailored versions of a document.

- Text-valued variables can be used in place of literal text any place that text is allowed within a document; generating a tailored instance of the document substitutes the variable's value for each reference to it.

The advantage of using any of the document processors (without TRF2) is that both creation of a document and its tailoring into instances occurs entirely within, and using the mechanisms of, the document processor.

Advantages of using Interleaf are:

- Individual graphical components can be selectively included or excluded in tailored versions of a document (given that they are named).
- Effectivity control can be used to tailor the content of linked documents, as a form of nested instantiation with attributes as parameters.

An advantage of using FrameMaker is that text formatting control can be included in the definition of a variable's value. TRF2 also supports this but lacks the ability to default formatting to that of the referencing context. Other packages support only the formatting defined by the referencing context.

The advantages of using TRF2 with Interleaf or FrameMaker are:

- The parameters of variation, analogous to attributes in Interleaf and variables in FrameMaker, are specifically defined and visible within the adaptable document. Attribute definitions are hidden within the document property sheet, variable definitions in a variables display form.
- The tailoring, inclusion or exclusion, and replication of text is explicit based on the parameters of variation, making their implications visible. Interleaf effectivity control is hidden with the document and component property sheets. Inclusion/exclusion in FrameMaker is limited to a simple user toggle, also hidden in a conditional text display form.
- Replication of components or fragments, with selective tailoring based on multivalued parameters, is supported.
- Interleaf supports color-coding by component type. By representing TRF2 constructs exclusively within a distinct component type, they can be color-coded to stand out from normal document content. FrameMaker provides similar support, but only for conditional text.
- TRF2 parameter values apply to a document as a whole, as do FrameMaker variables. Attribute values in Interleaf are associated with a page rather than with a component, a component type, or a document as a whole. This means that an attribute whose value applies to a full document must be assigned its value in at least one component on every page. When multiple components on a single page are assigned different values, it is not clear which takes precedence. Although an entire document can have a value associated with an attribute, its use is limited to deciding the inclusion or exclusion of the entire document in a book. Global assignments work only across an entire book (or document, if not resident within a book) but works only if every page contains at least one of the component types assigned the global value.

This page intentionally left blank.

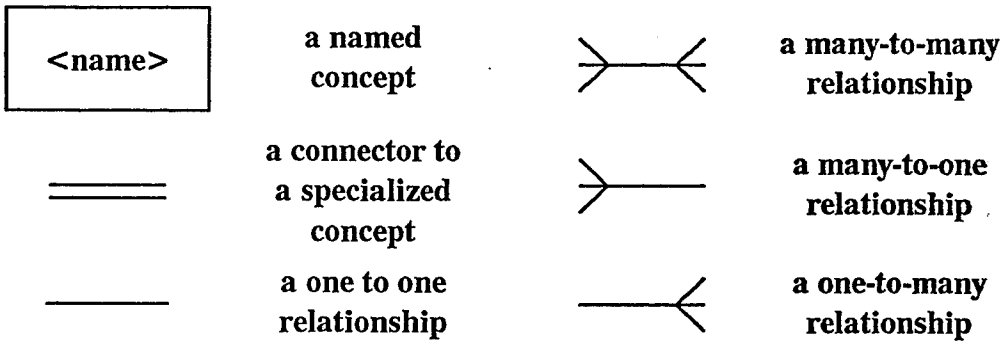
APPENDIX D. AN INFORMATION MODEL FOR LEVERAGED SYNTHESIS

The work products of a leveraged Synthesis process can take many forms. However, the essential information represented is the same, regardless of the specific form in which it is presented. This appendix formalizes the top-level information model that underlies the work products of a leveraged Synthesis process. The purpose of this model is to provide both a more generalized understanding of the information content of a Synthesis process and a basis from which tool vendors can create consistent and conforming tools. The version of the model presented here is influenced by the original Synthesis Reference Model (Software Productivity Consortium 1990a), the RSP Guidebook (Software Productivity Consortium 1993), and the Boeing/STARS logical model (Boeing 1994) that was created as the basis for a Domain Engineering automated environment which is currently under development as an extension of the SEE.

The following conventions are assumed:

- The purpose of the process is to produce the information identified by this model corresponding to particular business objectives and domain knowledge.
- Information versioning is outside this model. A valid instance of the model may contain multiple versions of any information, resulting from the repeated iteration of all or part of the process.
- The necessity of any information is defined by the process, not by this model. Any information may be known or unknown within a valid instance of the model, but the process may require that certain information be known for an instance of the model to be considered 'complete.'

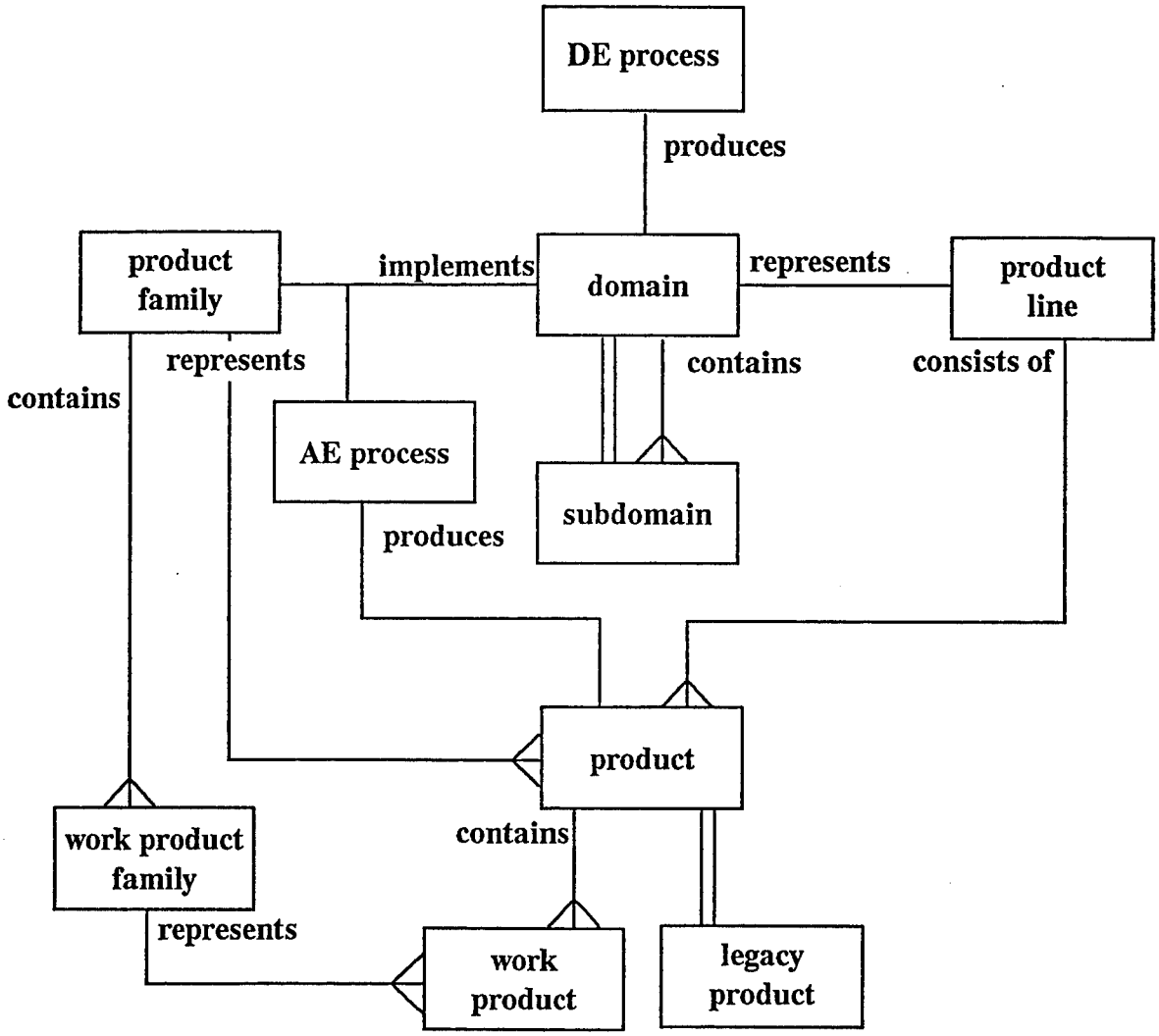
The following notations are used:



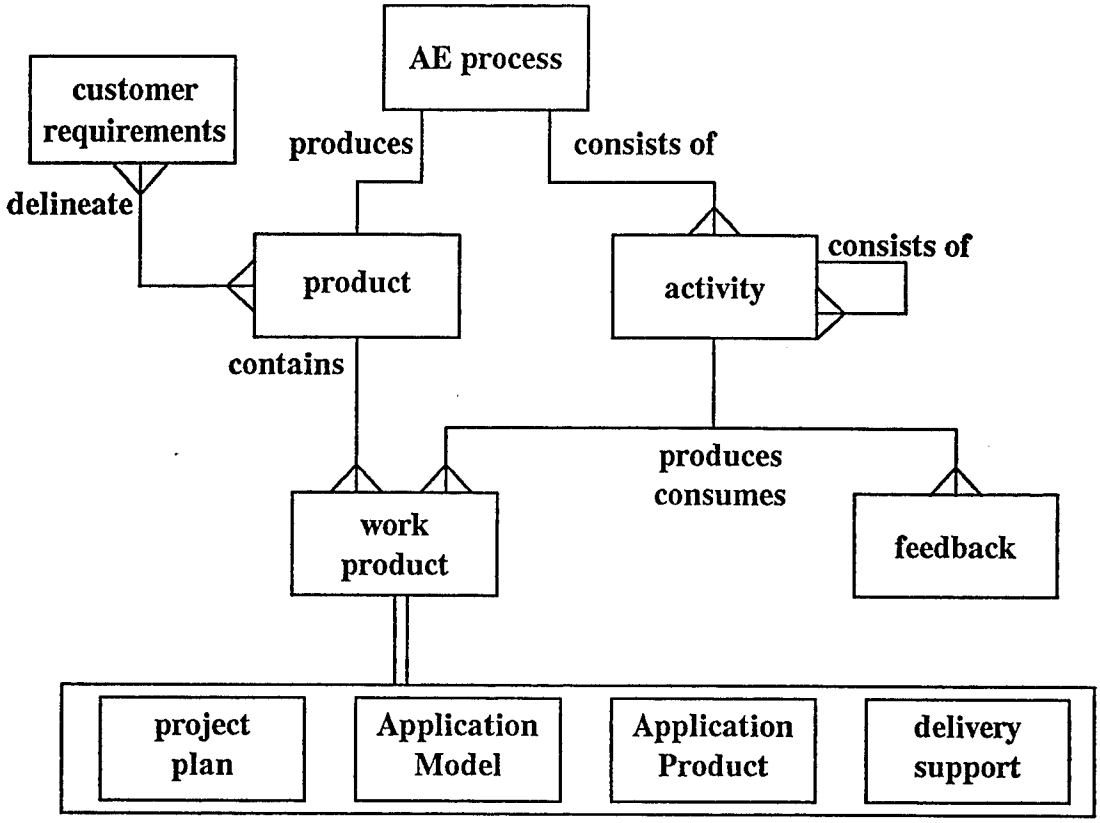
Relationships are labeled by verbs that denote meaning, with names placed nearest to the subject concept. Concepts are sometimes grouped within a box for notational convenience; for example, a

relationship shown from a concept to a box indicates that the relationship exists to each concept within the box.

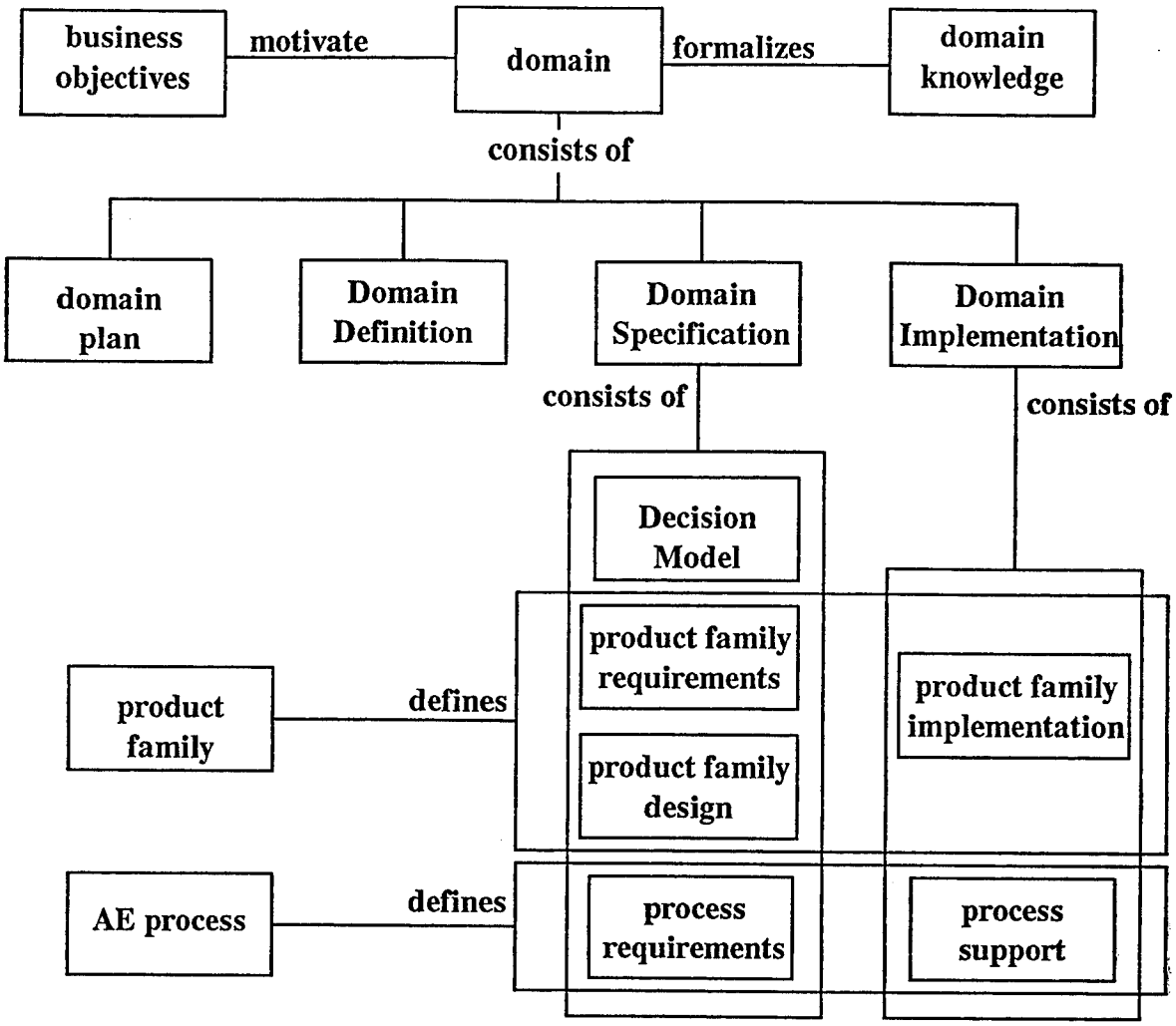
D.1 DOMAIN ENGINEERING PROCESS



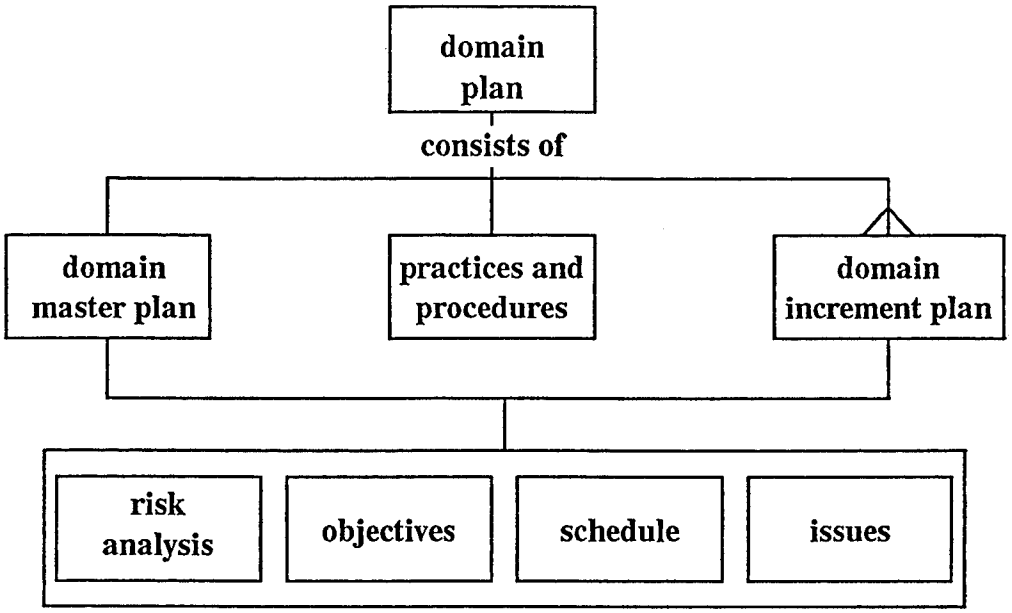
D.2 APPLICATION ENGINEERING PROCESS



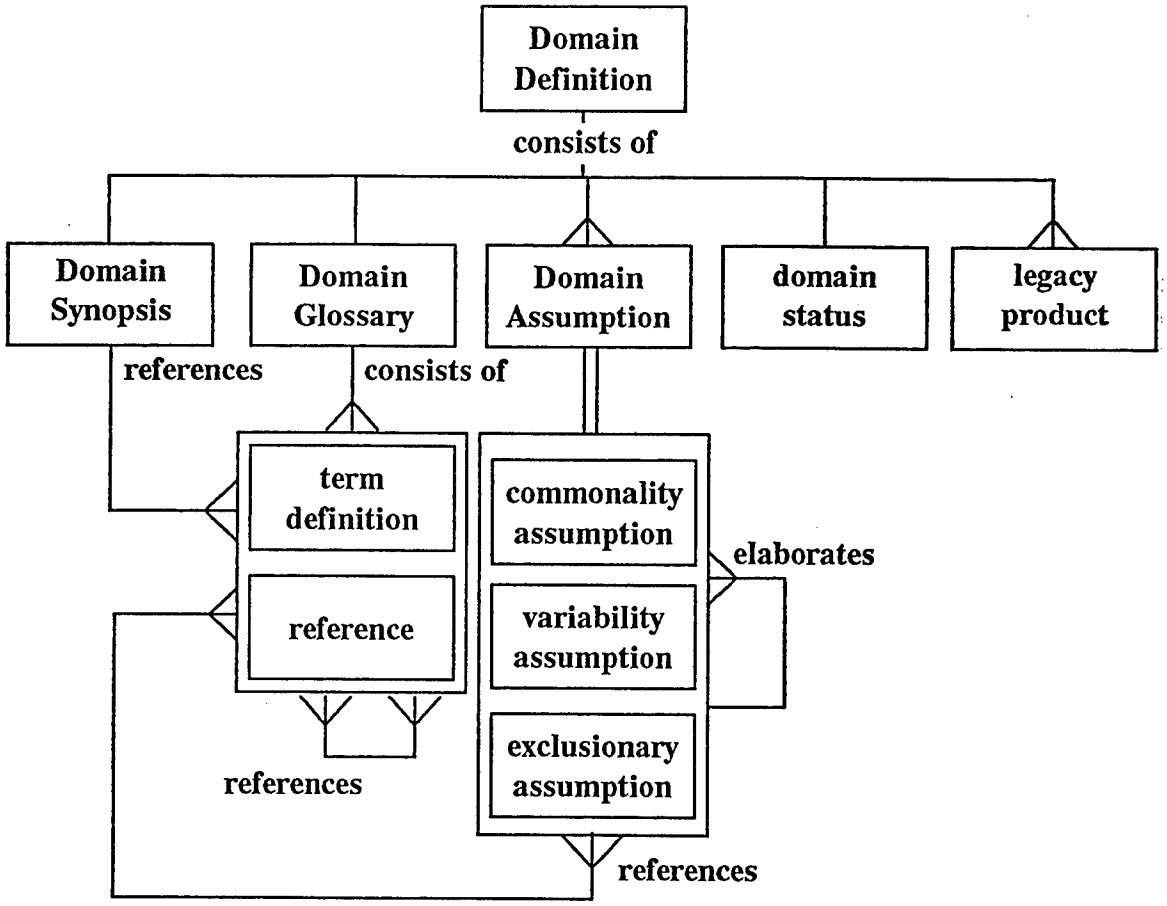
D.3 DOMAIN


business objectives
motivate
domain
formalizes
domain knowledge
consists of
domain plan
Domain Definition
Domain Specification
Domain Implementation
consists of
consists of
Decision Model
product family requirements
product family design
product family implementation
product family
defines
AE process
defines
process requirements
process support

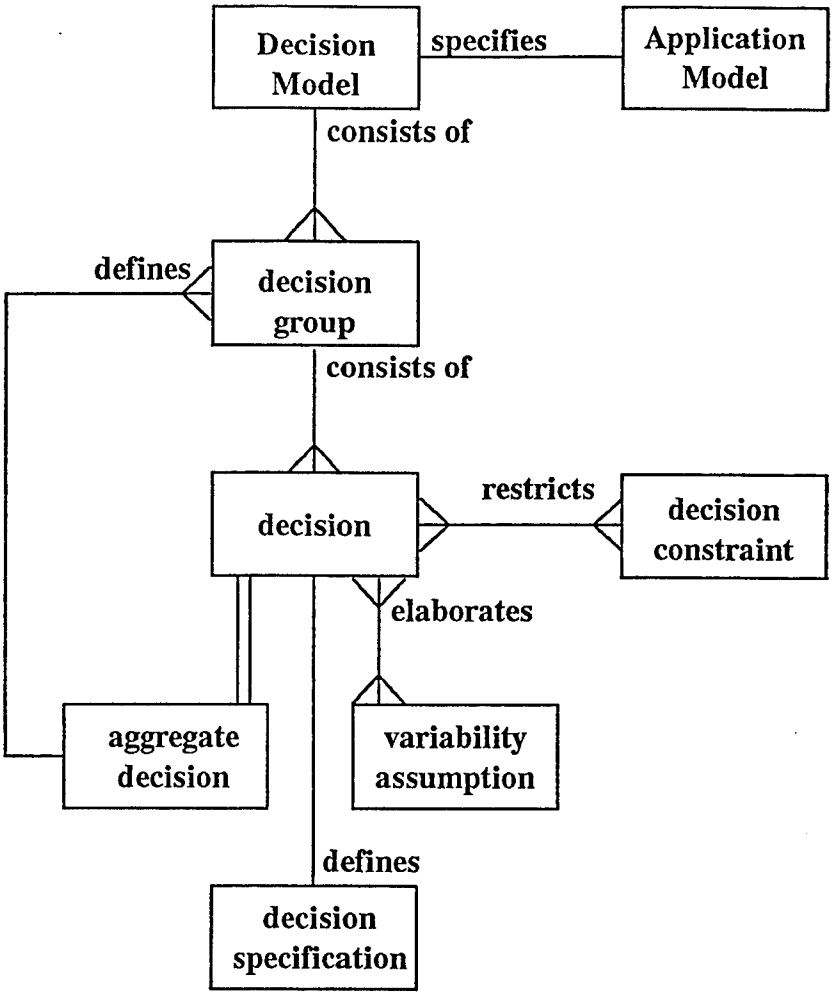
D.4 DOMAIN PLAN



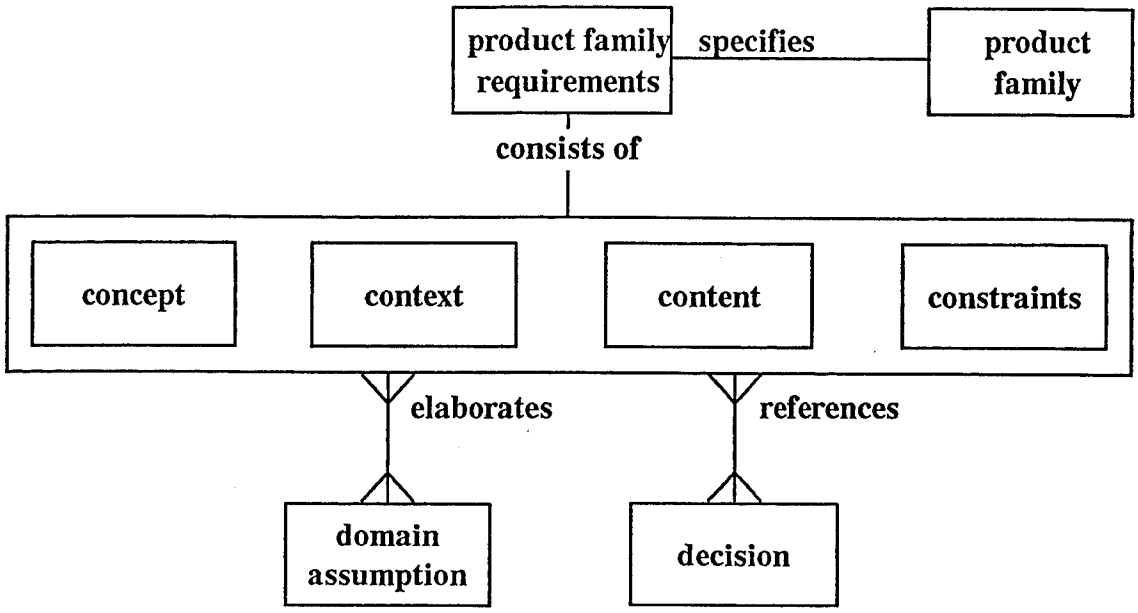
D.5 DOMAIN DEFINITION



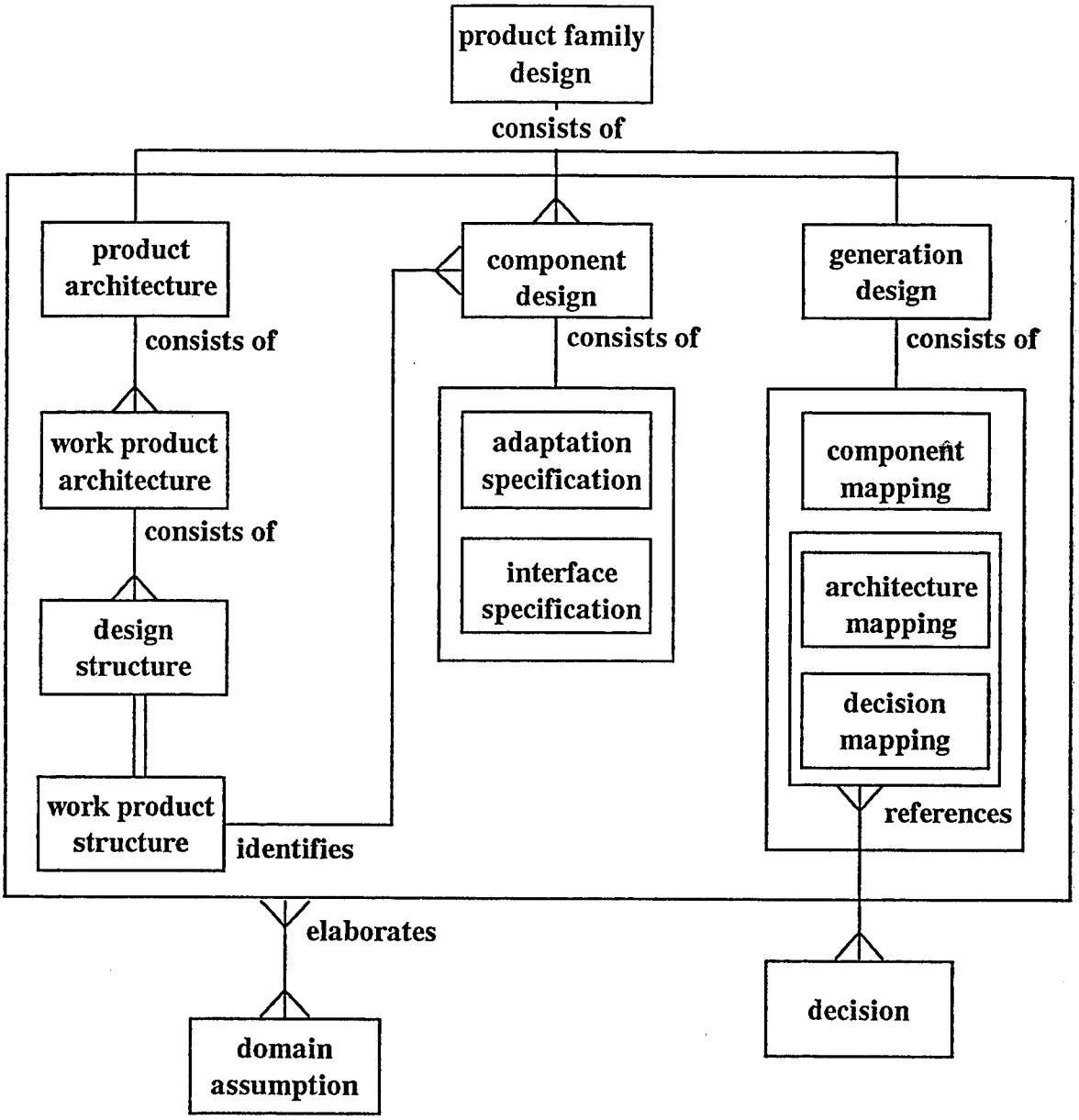
D.6 DECISION MODEL



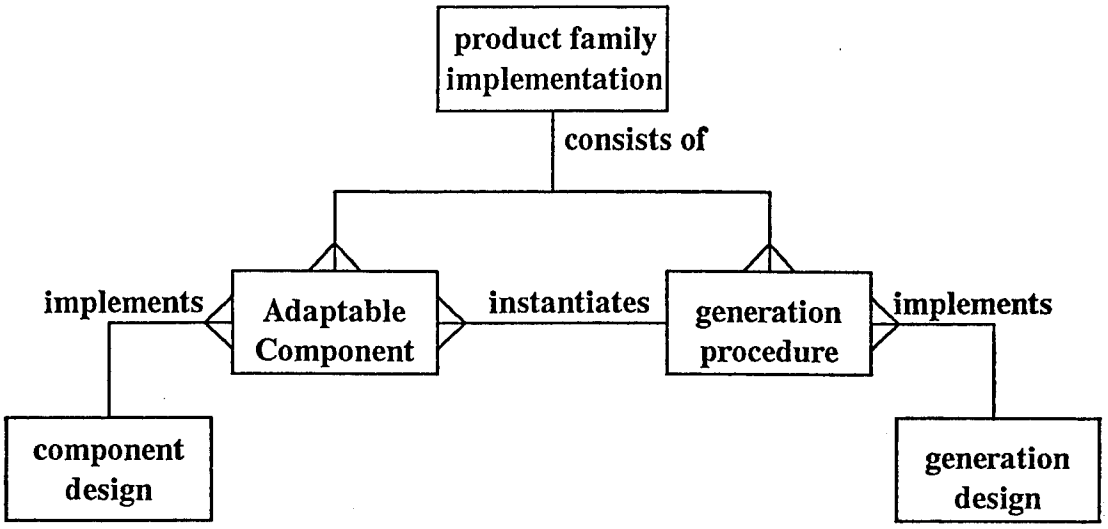
D.7 PRODUCT FAMILY REQUIREMENTS



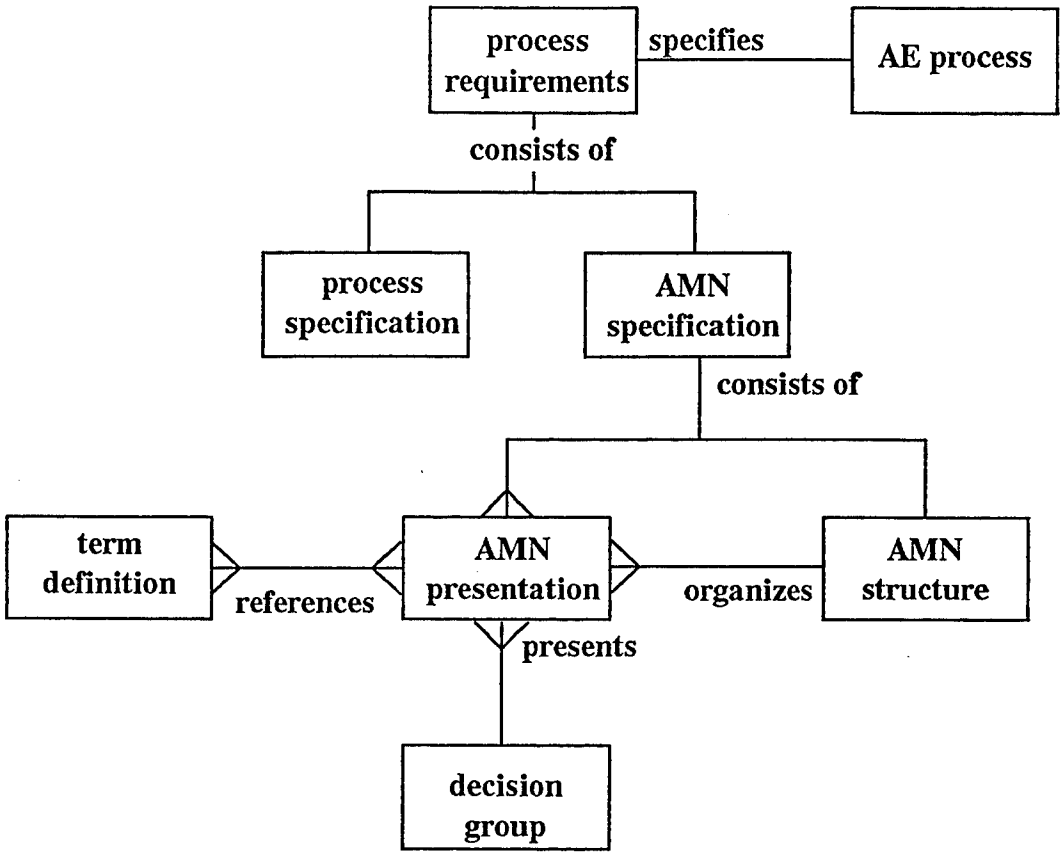
D.8 PRODUCT FAMILY DESIGN



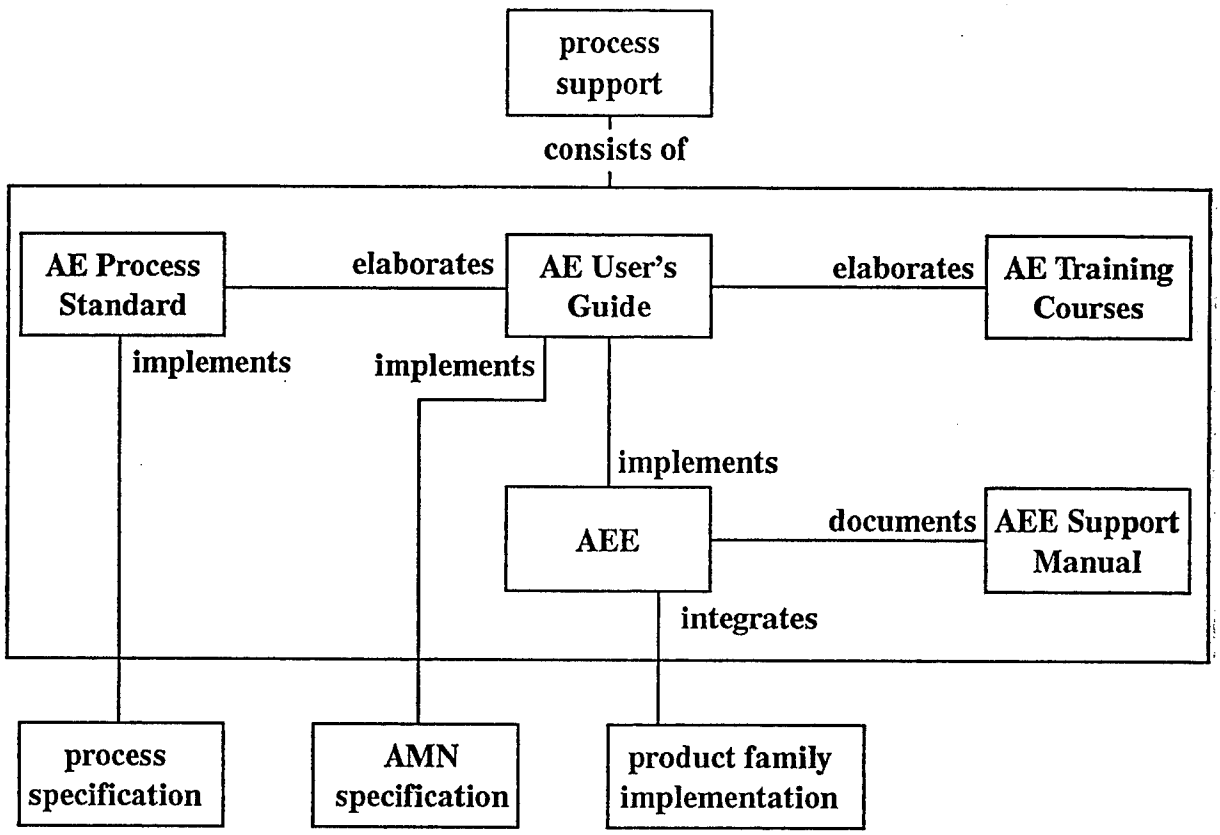
D.9 PRODUCT FAMILY IMPLEMENTATION



D.10 PROCESS REQUIREMENTS



D.11 PROCESS SUPPORT



APPENDIX E. REUSE LIBRARY TOOLS

Engineers and managers frequently associate reuse library tools with reuse practice. Reuse library tools provide mechanisms for the description, classification, storage, and retrieval of components (Software Productivity Consortium 1990b). Automated reuse library tools have a significant role in processes where application engineers opportunistically search for potentially reusable assets from a large collection then retrieve candidate assets for further analysis and adaptation. However, in a Synthesis process, opportunistic searching is replaced by mechanical identification and adaptation of assets based on engineering decisions; thus, the traditional reuse library tools have a more limited role in the process. Nevertheless, as demonstrated in the Boeing/STARS use of ROAMS (Section 3), reuse library tools can be used in a Synthesis process.

This appendix identifies existing reuse library tools, describes typical capabilities provided by these tools, and characterizes how reuse library tools can be used in a Synthesis process.

E.1 REUSE LIBRARY TOOLS

In the late 1980s, the reuse community viewed reuse libraries as the principal means to achieving higher levels of reuse. This view fostered the research, development, and ultimately the commercialization of many reuse library tools, such as the Consortium's Reuse Library Prototype (Software Productivity Consortium 1990b) that was adapted and commercialized by Atherton Technology. Later, the reuse community recognized that there were many other factors critical to successful reuse, such as process and organization, in which reuse libraries have a relatively small role (Software Productivity Consortium 1993).

Early in the STARS program, like the rest of the reuse community, the three STARS prime contractors placed much emphasis on software engineering environments in general and reuse library tools in particular. The STARS' effort resulted in the following reuse library tools:

- ROAMS, Boeing
- Reuse Library Framework, Unisys
- Asset Management System, Loral/SAIC

The STARS project also integrated their reuse library tools with their SEEs and are now placing more emphasis on integrating their engineering environments with their megaprogramming concept: process-driven, domain-specific, reuse-based, and technology-supported software engineering.

In line with the early thinking based on opportunistic searching of reuse libraries, several commercial products have come to market, including:

- InQuisiX, Software Productivity Solutions
- Reuse SoftBoard/Reuse Management System, Atherton Technology
- Reuse Library Toolset, EVB Software Engineering
- Reusable Search Expert, Westinghouse

CASE tool developers are also beginning to incorporate reuse library capabilities into their software development environments. Their current focus is on providing a common object repository to enable sharing of data among the various development tools. Although the focus of many of these tools is still on the development of a single system, they have the potential of being used to support multiple, if not family, systems development. An example is Logicore, which provides a common object repository for Logicon's I-CASE environment for the DoD (Bragg 1994).

Recently, due to the growing popularity of the World Wide Web, many organizations are using Internet discovery tools, such as Mosaic, WAIS, Gopher, and Archie, as a basis for implementing reuse libraries. The Repository Based Software Engineering program sponsored by NASA has implemented a reuse library tool based on Mosaic known as the Multimedia Oriented Repository Environment (MORE). A demonstration is available at URL http://rbse.jsc.nasa.gov:81/DEMO/.

E.2 TYPICAL REUSE LIBRARY TOOL CAPABILITIES

The reuse library tools identified in Section E.1 vary in their implementation, features, interfaces, platforms, but typically they provide the following capabilities:

- Storage of asset metadata (information about the assets) and, optionally, storage of the assets. Usually, various types of assets may be stored: for example, documents, code, and data. Typically, the tools allow the specification of user-defined (or domain-specific) asset types.
- Electronic access to the assets and their metadata, usually in a multiuser, local, and/or wide area network environment. Some tools also enable the protection of assets from unauthorized access.
- Configuration control over asset metadata and, optionally, the assets. This includes the unique identification of the assets, asset versioning, check in/check out of assets, status accounting and reporting, and auditing.
- Asset classification to facilitate searching. The tools support one or more classification schemes, such as enumerated (hierarchical), faceted, keyword, and uncontrolled (free text). Some tools also enable multiple classification schemes of the same type to present alternate views of the assets.
- Asset attributes and relationships definition. Attributes describe characteristics of the assets, for example, name, type, version, language, and size. Relationships define connections among different assets, for example, calls/called by, requires/required by, and uses/used by. Typically, the tools provide a base set of attribute and relationship types and enable the user to create additional (possibly domain-specific) attribute and relationship types.

- Searching, browsing, viewing, and retrieval of assets. Generally, the tools provide multiple mechanisms to help an application engineer find an asset, such as querying, keyword searches, and browsers that enable the engineer to follow the relationships between assets. Usually, the tools also provide a means to view the asset (if available electronically and text-based) before retrieving the asset. Retrieval may be done by electronically downloading the asset to the engineer's workstation or by placing an order for the asset.

E.3 USING REUSE LIBRARY TOOLS IN SYNTHESIS

Reuse library tools in Synthesis do not have the significant role that they have in opportunistic, library-based processes. In fact, there is greater potential use for reuse library tools in Domain Engineering than in Application Engineering, which is where they are used in library-based processes. Note that in the Rockwell case (Section 3.1), the CCSD did not use a reuse library tool. Nevertheless, reuse library tools can support a Synthesis process. This section describes the potential use of reuse library tools in both Application and Domain Engineering.

E.3.1 Reuse Library Tool Use in Application Engineering

In an opportunistic, library-based software process, reuse library tools are principally used by application engineers to identify, select, and retrieve reusable assets. However, in a Synthesis process the identification and selection of reusable assets is driven by the Decision Model and the architecture and component mappings. Thus, in an Application Engineering context of a Synthesis process, the searching and browsing features of a reuse library tool are not essential. What remains essential are storage, configuration management, and retrieval capabilities, which a reuse library tool typically provides.

The Boeing/Navy STARS case (Section 3.2) illustrates how Boeing used ROAMS in a Synthesis Application Engineering process. In this case, ROAMS provided the storage mechanism for reusable assets. Kamel interacted with the application engineer in constructing an Application Model. Based on the Application Model, Kamel generated retrieval and adaptation directives. The Process Engine then supplied the retrieval instructions to ROAMS to obtain the asset, which was then adapted by Kamel. In this process, the application engineer had little to no direct contact with the reuse library tool.

E.3.2 Reuse Library Tool Use in Domain Engineering

A suitable use for reuse library tools in Domain Engineering is in the inventorying, storage, and retrieval of Legacy Products. In this mode of operation, all of the typical capabilities provided by a reuse library tool are useful. The architectural and associative relationships between elements of a system can be captured by storing the elements as assets and creating asset relationships corresponding to the architectural relationships. Likewise, asset relationships can be created to connect similar assets. Then, in the course of Domain Specification and Implementation, a domain engineer can use the reuse library tool to search for and retrieve potential legacy assets for reuse.

A somewhat lesser use for a reuse library tool in Domain Engineering is to manage a repository of domain work products, such as the Domain Synopsis, Domain Assumptions, Specifications, and Adaptable Components. Domain engineers could use the tool to maintain configuration control over domain work products as well as capture relationships between the products. Note, however, that these capabilities could also be provided effectively by a good configuration management tool.

This page intentionally left blank.

APPENDIX F. TOOL CONTACT INFORMATION

This appendix enumerates the commercial tools mentioned in this report and identifies platforms and vendor name and telephone for each. Inclusion in this list does not constitute a recommendation of a tool nor does failure to reference a tool indicate that it is any less viable for potential use within a leveraged Synthesis process than any similar tools listed. A comparison of all similar tools should be conducted as part of the selection of a tool for any purpose. Information given here is based partially on product descriptions provided in CASE Trends (1993) and in Ziff (1994).

For each referenced tool, its availability on popular platforms is identified. Platforms include PC (any of Microsoft DOS, Windows, or NT), UNIX (any one or more UNIX platforms), Mac (Apple Macintosh), and VMS (DEC). Some tools are also available on other platforms as well, particularly mainframes. Vendors should be consulted for current and comprehensive information on their tools.

Tool	Platform	Vendor
AllChange	PC, UNIX	Intasoft Ltd., UK +44-392-217670
Amadeus	UNIX	Amadeus Software Research (617) 326-9339
ART	UNIX, VMS	Inference Corp. (800) 322-9923
Bachman/Analyst Capture	OS/2	Bachman Information Systems, Inc. (800) 222-4626
Borland C++	PC	Borland International, Inc. (800) 233-2444
BPwin	PC	Logic Works, Inc. (800) 783-7946
CaseWare/CM	UNIX	CaseWare, Inc. (714) 453-2200
CaseWare/PT	UNIX	CaseWare, Inc. (714) 453-2200
CA-SuperProject	PC, VMS	Computer Associates International, Inc. (800) 225-5224
CCC/Manager	PC, UNIX, VMS, OS/2	Softool Corp. (800) 723-0696

Tool	Platform	Vendor
CCC/Pro	UNIX, VMS	Softool Corp. (800) 723-0696
CDD	VMS	Digital Equipment Corp. (800) 777-4343
ClearCase	PC, UNIX	Atria Software, Inc. (508) 650-5100
ClearQuality	PC, UNIX, Mac	Clarify, Inc. (408) 428-2000
CMS and MMS	VMS	Digital Equipment Corp. (800) 344-4825
Cohesion	VMS	Digital Equipment Corp. (800) 344-4825
DB2	OS/2, UNIX	IBM (800) 426-3333
DECPlan	VMS	Digital Equipment Corp. (800) 344-4825
DEC Rdb	VMS	Digital Equipment Corp. (800) 344-4825
Design/IDEF	Mac	Meta Software Corp. (800) 227-4106
Eiffel 3	Mac, UNIX, VMS	Interactive Software Engineering, Inc. (805) 685-1006
FileMaker Pro	Mac, PC	Claris Corp. (800) 544-8554
4th Dimension	Mac	ACI US, Inc. (800) 384-0010
FrameMaker	UNIX, OS/2	Frame Technology Corp. (800) 843-7263
Galaxy Application Environment	PC, Mac, OS/2, UNIX, VMS	Visix Software, Inc. (800) 832-8668
GemStone	UNIX, PC, Mac, VMS	Servio Corp. (800) 243-9369
Information Engineering Facility	MVS, UNIX, VMS, OS/2, PC, Mac	Texas Instruments, Inc. (800) 336-5236

Tool	Platform	Vendor
InQuisiX	UNIX	Software Productivity Solutions, Inc. (407) 984-3370
Interleaf	UNIX	Interleaf, Inc. (800) 955-5323
IslandWrite	UNIX	Island Software Corp. (800) 255-4499
Logiscope	UNIX, VMS	Logiscope Technologies, Inc. (214) 241-6595
Lotus 1-2-3	UNIX, Mac, PC, VMS, OS/2	Lotus Development Corp. (800) 343-5414
Lotus Notes	PC, UNIX, OS/2	Lotus Development Corp. (800) 343-5414
MacProject Pro	Mac	Claris Corp. (800) 325-2747
MatrixX	UNIX	Integrated Systems, Inc. (408) 980-1500
McCabe Instrumentation Tool	PC, UNIX, VMS	McCabe & Associates, Inc. (800) 638-6316
Microsoft Access	PC	Microsoft Corp. (800) 426-9400
Microsoft Excel	PC, Mac	Microsoft Corp. (800) 426-9400
Microsoft Project	PC, Mac	Microsoft Corp. (800) 426-9400
Microsoft Visual C++	PC	Microsoft Corp. (800) 426-9400
Microsoft Word	PC, Mac	Microsoft Corp. (800) 426-9400
NextStep	UNIX	NeXT Computer, Inc. (800) 879-6398
ObjecTime	UNIX	ObjecTime, Ltd. (800) 567-8463
ObjectMaker	UNIX, VMS, Mac, OS/2, PC	Mark V Systems, Ltd. (818) 995-7671

Tool	Platform	Vendor
Objectory	UNIX, PC, OS/2	Objectory Corp. (203) 625-7250
ObjectStore	UNIX, OS/2	Object Design, Inc. (800) 962-9620
OMTool	UNIX, PC	Martin Marietta Corp. (800) 438-7246
Ontos DB	UNIX, OS/2	ONTOS, Inc. (800) 388-7110
Ontos Studio	UNIX, OS/2	ONTOS, Inc. (800) 388-7110
Open Interface	PC, Mac, OS/2, VMS, UNIX	Neuron Data, Inc. (800) 876-4900
OpenSelect	PC	Rational Software Corp. (originally Meridian Software Systems) (800) 433-5444
Oracle	UNIX, OS/2, Mac	Oracle Corp. (800) 672-2531
Paradox	PC	Borland International, Inc. (800) 233-2444
PowerBuilder	Mac, UNIX	Powersoft Corp. (800) 273-2841
PROCESS WEAVER	UNIX, PC	Cap Gemini America (213) 291-7804
Purify	UNIX	Pure Software, Inc. (800) 353-7873
PVCS	PC, OS/2, UNIX	Intersolv, Inc. (800) 547-4000
QualTEAM	Ultrix, UNIX, IBM RS/6000, Sun, VAX	Scopus Technology, Inc. (510) 428-0500
Rational Rose	UNIX, OS/2	Rational Software Corp. (800) 767-3237
RCE	PC, UNIX	Xcc Software Technology Transfer, Germany +49-721-616474
RDD-100	PC, Mac, UNIX	Ascent Logic Corp. (800) 654-4733

Tool	Platform	Vendor
Refine	UNIX	Reasoning Systems (415) 494-6201
Reusable Search Expert	Sun, VMS	Westinghouse, Inc. (800) 742-4802
Reuse Library Toolset	PC, UNIX	EVB Software Engineering, Inc. (800) 877-1815
Reuse Management System	UNIX	Atherton Technologies, Inc. (800) 984-7233
Smalltalk/V	PC, Mac, OS/2	Digitalk, Inc. (800) 531-2344
SNAP	UNIX, VMS, PC	Template Software, Inc. (703) 318-1000
SoftBench	UNIX	Hewlett-Packard Co. (800) 752-0900
Software Through Pictures Structured Environment	UNIX	Interactive Development Environments (800) 888-4331
Statemate	UNIX, VMS	i-Logix, Inc. (508) 682-2100
SunPro Workshop	UNIX	Sun Microsystems, Inc. (800) 926-6620
Sybase Open Client; Sybase Open Server	UNIX, VMS, OS/2	Sybase, Inc. (800) 879-2273
Symantec C++	Mac, PC	Symantec Corp. (800) 441-7234
SynerVision	UNIX, Sun	Hewlett-Packard Co. (800) 858-8867
Team*work* C/Rev	OS/2, UNIX, VMS	Cadre Technologies, Inc. (800) 743-2273
Teamwork for Structured Methods	UNIX, VMS, OS/2	Cadre Technologies, Inc. (800) 743-2273
Time Line	PC	Symantec Corp. (800) 441-7234
Versant ODBMS	UNIX, OS/2	Versant Object Technology Corp. (800) 837-7268

Tool	Platform	Vendor
Virtual Software Factory	Ultrix, VAX, Sun, IBM RS/6000	Virtual Software Factory Ltd. (703) 318-1190
Visual Basic	PC	Microsoft Corp. (800) 426-9400
VisualWorks (Smalltalk)	Mac, UNIX, OS/2	ParcPlace Systems, Inc. (800) 759-7272
WordPerfect	PC, Mac, UNIX	WordPerfect Corporation (800) 451-5151
XVT	PC, Mac, OS/2, UNIX	XVT Software, Inc. (800) 678-7988

LIST OF ABBREVIATIONS AND ACRONYMS

AAA	Agents, Artifacts, and Activities
ADARTS	Ada-based Design Approach for Real-Time Systems
AE	Application Engineering
AEE	Application Engineering Environment
AMN	Application Modeling Environment
ARPA	Advanced Research Projects Agency
ATIS	A Tool Integration Standard
AVTS	Air Vehicle Training System
CASE	computer-aided software engineering
CCSD	Command and Control Systems Division (Rockwell)
CLIPS	C Language Integrated Production System
CM	configuration management
CORBA	Common Object Request Broker Architecture
CoRE	Consortium Requirements Engineering
COTS	commercial off-the-shelf
DARTS	Domain Architecture for Reusable Training Systems
DBMS	database management system
DE	Domain Engineering
DEC	Digital Equipment Corporation
ESP	Evolutionary Spiral Process
FIT	Flight Instrument Trainer
I-CASE	integrated computer-aided software engineering

IR&D	internal research and development
KAMEL	Knowledge-based Application Model EvaLuation
MORE	Multimedia Oriented Respository Environment
NAWCTSD	Naval Air Warfare Center Training Systems Division
OMG	Object Management Group
PDL	Program Design Language
PERT	Performance Evaluation Review Technique
POSIX	Portable Operating System for Computer Environments
ROAMS	Reusable Object Access and Management System
RSP	Reuse-Driven Software Processes
RTSA	Real-Time Structured Analysis
SEE	Software Engineering Environment
SRS	Software Requirements Specification
STARS	Software Technology for Adaptable, Reliable Systems
WAIS	Wide Area Information System

GLOSSARY

These definitions are taken from the *Reuse-Driven Software Processes Guidebook* (Software Productivity Consortium 1993). Refer to that document for additional information.

Activity	A step of a process for producing and/or evaluating work products to satisfy objective(s) supporting that process. An activity comprises other steps.
Application Engineering	An iterative process for the design and development of a product that satisfies specified customer requirements. Its work products are an Application Model and an Application Product.
Application Model	A set of resolved requirements and engineering decisions, as specified by a Decision Model, that (partially) determine an instance of a family of systems.
Application Product	Software artifacts, including code and documentation, produced by Application Engineering to satisfy customer requirements.
Commonality	A characteristic of a domain that corresponds to a similarity among members of the associated family of systems. See Variability.
Decision Model	The Domain Engineering work product that defines the abstract form (concepts, decisions, and structure) of an Application Model.
Domain	A product family and an associated production process supporting a product line.
Domain Analysis (Activity)	The Domain Engineering activity in which domain knowledge is studied and formalized. The expertise in a business area is formalized to create standards for problem descriptions and corresponding solutions.
Domain Engineering	An iterative process for the design and development of (1) a product family and (2) an Application Engineering process for producing members of that family.

Domain Implementation (Activity)	The Domain Engineering activity that creates support for Application Engineering projects in the form of a Domain Implementation.
Domain Management (Activity)	The Domain Engineering activity that plans, monitors, and controls the activities and resources of a Domain Engineering organization and which coordinates domain development and evolution with client Application Engineering projects.
Family	A set of things that have enough in common that it pays to consider their common characteristics before noting specific properties of instances.
Method	Guidance and criteria that prescribe a systematic, repeatable technique for performing an activity.
Methodology	An integrated body of principles, practices, and methods that prescribe the proper performance of a process.
Process	A (partially) ordered set of steps, intended to accomplish specified objective(s).
Product	The aggregation of all work products resulting from a process or activity.
Product family	A representation of a set of products that are characterized by specified commonalities and variabilities. See Family.
Product line	A collection of (existing and potential) products that address a designated business area.
Project Support (Activity)	The Domain Engineering activity that validates a Domain Implementation, delivers it to Application Engineering projects, and supports its use.
Synthesis	A methodology for the construction of software systems as instances of a family of systems that have similar descriptions. Its primary distinguishing features are: • Formalization of domains as families of systems that share many common features, but which also vary in well-defined ways

Synthesis (cont.)	• System building reduced to resolution of requirements and engineering decisions that represent the variations characteristic of a domain • Reuse of software artifacts through mechanical adaptation of components to satisfy requirements and engineering decisions • Model-based analyses of described systems to help understand the implications of system-building decisions and evaluate alternatives
System	A collection of hardware, software, and people that operate together to accomplish a mission.
Variability	A characteristic of a domain that corresponds to features that distinguish among members of the associated family of systems. See Commonality.

This page intentionally left blank.

REFERENCES

Boeing 1994 — Baselined Logical Model as of June 17 (unpublished). Seattle, Washington: Boeing Defense & Space Group.

Bragg, Thomas 1994 — A CASE for Defense: The Defense Department's 10-Year Integrated CASE Project. *American Programmer* 7, 7:16–22.

CASE Trends 1993 — *CASE Trends* 5, 4, Shrewsbury, MA: Software Productivity Group, Inc.

Eaton, Dave 1994a — *Configuration Management Tools Summary*, comp.software.config–mgmt FAQ.

1994b — *Problem Management Tools Summary*, comp.software.config–mgmt FAQ.

Hausler, P. A., R. C. Linger, and C. J. Trammell 1994 — Adopting Cleanroom Software Engineering With a Phased Approach. *IBM Systems Journal* 33, 1:89–109.

O'Connor, James, Catharine Mansour, Jerri Turner-Harris, and Grady Campbell 1994 — Reuse in Command-and-Control Systems. *IEEE Software* 11:70–79.

Software Productivity Consortium 1990a — *Synthesis Methodology Reference Model*, SYNTHESIS-REF-MODEL-90047-MC, version 01.00.03. Herndon, Virginia: Software Productivity Consortium.

1990b — *Usage Scenario for the Reuse Library Toolset*, EX_US_RLT-90052-MC. Herndon, Virginia: Software Productivity Consortium.

1991 — *TRF2 Metaprogramming Tool User Guide*, SPC-91132-MC, version 01.00.02. Herndon, Virginia: Software Productivity Consortium.

1993 — *Reuse-Driven Software Processes Guidebook*, SPC-92019-CMC, version 02.00.03. Herndon, Virginia: Software Productivity Consortium.

STARS 1993 — *Integrated SEE Description SEE Reference Manual*, D613-61016-1 (Draft). Seattle, Washington: Boeing Defense & Space Group.

1994a — *Enabling AAA Runtime Support Using Denali.* Seattle, Washington: Software Technology for Adaptable Reliable Systems.

1994b — *Experience Reports — The Navy/STARS Demonstration Project.* Seattle, Washington: Software Technology for Adaptable Reliable Systems.

STSC 1993 — *Project Management Technology Report.* Hill Air Force Base, Utah: Software Technology Support Center.

1994 — *Process Technologies Method and Tool Report.* Hill Air Force Base, Utah: Software Technology Support Center.

Ziff 1994 — *Computer Select.* New York, New York: Ziff Communications Co.