

NAVAL POSTGRADUATE SCHOOL

Monterey, California



DISSERTATION

**HETEROGENEOUS SOFTWARE SYSTEM
INTEROPERABILITY THROUGH COMPUTER-AIDED
RESOLUTION OF MODELING DIFFERENCES**

by

Paul E. Young

June 2002

Dissertation Supervisor:

Luqi

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2002	3. REPORT TYPE AND DATES COVERED Dissertation	
4. TITLE AND SUBTITLE: Heterogeneous Software System Interoperability Through Computer-Aided Resolution of Modeling Differences			5. FUNDING NUMBERS	
6. AUTHOR(S) Paul E. Young				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Space and Naval Warfare Systems Center- San Diego San Diego, CA 92152-5031			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) Meeting future system requirements by integrating existing stand-alone systems is attracting renewed interest. Computer communications advances, functional similarities in related systems, and enhanced information description mechanisms suggest that improved capabilities may be possible; but full realization of this potential can only be achieved if stand-alone systems are fully interoperable. Interoperability among independently developed heterogeneous systems is difficult to achieve: systems often have different architectures, different hardware platforms, different operating systems, different host languages and different data models. The Object-Oriented Method for Interoperability (OOMI) introduced in this dissertation resolves modeling differences in a federation of independently developed heterogeneous systems, thus enabling system interoperation. First a model of the information and operations shared among systems, termed a Federation Interoperability Object Model (FIOM), is defined. Construction of the FIOM is done prior to run-time with the assistance of a specialized toolset, the OOMI Integrated Development Environment (OOMI IDE). Then at runtime OOMI Translators utilize the FIOM to automatically resolve differences in exchanged information and in inter-system operation signatures.				
14. SUBJECT TERMS Interoperability, Model Correlation, Heterogeneous Software Systems, XML, Data Binding, Modeling Difference Resolution			15. NUMBER OF PAGES 306	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**HETEROGENEOUS SOFTWARE SYSTEM INTEROPERABILITY THROUGH
COMPUTER-AIDED RESOLUTION OF MODELING DIFFERENCES**

Paul E. Young
Captain, United States Navy
B.S., University of Mississippi, 1977
M.S., University of Mississippi, 1985
M.S., Naval Postgraduate School, 2001

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN SOFTWARE ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
June 2002**

Author:

Paul E. Young

Approved by:

Luqi
Professor of Computer Science
Dissertation Supervisor

Valdis Berzins
Professor of Computer Science

William Kemple
Associate Professor of Command,
Control and Communications

Edmund Freeman
Science Applications
International Corporation

Jun Ge
Research Associate

Richard Riehle
Instructor of Computer Science

Approved by:

Chris Eagle, Chair, Department of Computer Science

Approved by:

Carson K. Eoyang, Associate Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Meeting future system requirements by integrating existing stand-alone systems is attracting renewed interest. Computer communications advances, functional similarities in related systems, and enhanced information description mechanisms suggest that improved capabilities may be possible; but full realization of this potential can only be achieved if stand-alone systems are fully interoperable. Interoperability among independently developed heterogeneous systems is difficult to achieve: systems often have different architectures, different hardware platforms, different operating systems, different host languages and different data models.

The Object-Oriented Method for Interoperability (OOMI) introduced in this dissertation resolves modeling differences in a federation of independently developed heterogeneous systems, thus enabling system interoperation. First a model of the information and operations shared among systems, termed a Federation Interoperability Object Model (FIOM), is defined. Construction of the FIOM is done prior to run-time with the assistance of a specialized toolset, the OOMI Integrated Development Environment (OOMI IDE). Then at runtime OOMI translators utilize the FIOM to automatically resolve differences in exchanged information and in inter-system operation signatures.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	QUEST FOR SYSTEM INTEROPERABILITY.....	1
B.	LIMITATIONS WITH CURRENT APPROACHES TOWARD ACHIEVING INTEROPERABILITY.....	4
C.	RESEARCH QUESTION AND HYPOTHESIS.....	5
D.	OBJECT-ORIENTED METHOD FOR INTEROPERABILITY (OOMI) OVERVIEW.....	6
E.	CONTRIBUTIONS PROVIDED BY THIS DISSERTATION.....	7
F.	IMPACT AND LONG-TERM SIGNIFICANCE OF DISSERTATION’S CONTRIBUTIONS.....	7
G.	DISSERTATION ORGANIZATION.....	8
II.	SURVEY OF PREVIOUS WORK ON ACHIEVING INTEROPERABILITY AMONG INDEPENDENTLY DEVELOPED SYSTEMS.....	11
A.	MODELING DIFFERENCES AMONG SYSTEMS.....	11
1.	Causes of Modeling Differences.....	11
2.	Kinds of Modeling Differences.....	11
a.	<i>Heterogeneity of Hardware and Operating Systems.....</i>	<i>12</i>
b.	<i>Heterogeneity of Organizational Models.....</i>	<i>12</i>
c.	<i>Heterogeneity of Structure.....</i>	<i>13</i>
d.	<i>Heterogeneity of Presentation.....</i>	<i>13</i>
e.	<i>Heterogeneity of Meaning.....</i>	<i>14</i>
f.	<i>Heterogeneity of Scope.....</i>	<i>14</i>
g.	<i>Heterogeneity of Level of Abstraction.....</i>	<i>15</i>
h.	<i>Heterogeneity of Temporal Validity.....</i>	<i>15</i>
B.	CRITERIA FOR EVALUATING INTEROPERABILITY APPROACHES.....	15
1.	Types of Heterogeneity Addressed.....	15
2.	Capability for Application of Computer Aid for Model Correlation.....	16
3.	Required Knowledge of Remote Operations.....	16
4.	Required Modification to Existing System.....	17
5.	Translation Methodology.....	17
6.	Capability for Application of Computer Aid for Translation Development.....	17
7.	Support for Federation Extensibility.....	18
8.	Information Exchange Versus Joint Task Execution.....	18
C.	APPROACHES FOR ACHIEVING INTEROPERABILITY AMONG HETEROGENEOUS SYSTEMS.....	19
1.	Common Object Request Broker Architecture (CORBA).....	19
a.	<i>CORBA Overview.....</i>	<i>19</i>

	b.	<i>CORBA Architecture Overview</i>	20
	c.	<i>Evaluation of Interoperability Approach</i>	23
2.		COM, DCOM, COM+	30
	a.	<i>Component Object Model (COM)</i>	30
	b.	<i>Distributed Component Object Model (DCOM)</i>	35
	c.	<i>Component Object Model Plus (COM+)</i>	37
	d.	<i>Evaluation of Interoperability Approach</i>	38
3.		Java 2 Enterprise Edition (J2EE).....	44
	a.	<i>J2EE Overview</i>	44
	b.	<i>Evaluation of Interoperability Approach</i>	49
4.		SeeBeyond Integration Suite	53
	a.	<i>e*GateTM Integrator Overview</i>	53
	b.	<i>e*Index Global Identifier Overview</i>	58
	c.	<i>Evaluation of Interoperability Approach</i>	60
5.		The High Level Architecture for Modeling and Simulation (HLA).....	64
	a.	<i>HLA Elements</i>	64
	b.	<i>HLA Specification</i>	65
	c.	<i>Evaluation of Interoperability Approach</i>	68
6.		eXtensible Markup Language (XML).....	73
	a.	<i>XML basics</i>	74
	b.	<i>Constraining Content</i>	75
	c.	<i>Programmatic Access</i>	76
	d.	<i>Translations</i>	77
	e.	<i>XML Data-Binding</i>	77
	f.	<i>Evaluation of Interoperability Approach</i>	80
D.		SUMMARY.....	84
III.		THEORETICAL FOUNDATION FOR COMPONENT SYSTEM OBJECT CORRELATION.....	89
A.		CORRELATION MEASURES OF EFFECTIVENESS	89
B.		DATA CORRELATION METHODS	90
	1.	Classical Approaches	91
		a. <i>Browsing</i>	91
		b. <i>Keyword Matching</i>	91
		c. <i>Multi-Attribute Search</i>	92
		d. <i>Classical Approach Applicability to Interoperability Correlation Problem</i>	94
	2.	Formal Specifications.....	94
		a. <i>Syntax Based Approach</i>	95
		b. <i>Semantics Based Approach</i>	96
		c. <i>Approach Using Component Syntax and Semantics</i>	98
		d. <i>Formal Specifications Applicability to Interoperability Correlation Problem</i>	112
	3.	Artificial Intelligence Approaches	113
		a. <i>Natural Language Techniques</i>	113
		b. <i>Neural Networks</i>	118

	c.	<i>Applicability of Artificial Intelligence Approaches to Interoperability Correlation Problem</i>	122
C.	SUMMARY.....		123
IV.	OBJECT-ORIENTED METHOD FOR INTEROPERABILITY (OOMI).....		125
A.	INTRODUCTION.....		125
B.	METHODS FOR RESOLVING HETEROGENEITY AMONG SYSTEMS.....		125
C.	OBJECT-ORIENTED METHOD FOR INTEROPERABILITY (OOMI).....		128
	1.	Federation Interoperability Object Model (FIOM).....	130
	a.	<i>Categories of Modeling Differences</i>	130
	b.	<i>FIOM Composition</i>	135
	2.	OOMI Integrated Development Environment (IDE)	146
	3.	OOMI Translator.....	148
D.	SUMMARY.....		150
V.	OBJECT-ORIENTED METHOD FOR INTEROPERABILITY INTEGRATED DEVELOPMENT ENVIRONMENT (OOMI IDE)		153
A.	OOMI IDE PURPOSE.....		153
B.	OOMI IDE DEVELOPMENT CONSIDERATIONS		154
C.	FEDERATION INTEROPERABILITY OBJECT MODEL (FIOM) CONSTRUCTION PROCESS.....		155
	1.	Add Component System External Interface.....	157
	2.	Manage Federation Entities (FEs).....	158
	3.	Register Component Class Representation (CCR)	159
	a.	<i>Finding FE Corresponding To CCR Being Registered</i>	159
	b.	<i>Modifying FIOM to Provide Required Correspondence Between CCR and FCR Schema</i>	160
	c.	<i>Adding CCR to FEV Whose FCR Schema Exhibits a One-To-One Correspondence with the CCR Schema Being Registered</i>	161
	d.	<i>Adding Translations Between Component And Federation Class Representations Of Real-World Entity</i> ...	161
	4.	Update Federation Ontology.....	162
	5.	Generate System-Specific Translator Information.....	163
D.	OOMI IDE PROTOTYPE		163
	1.	User Interface	165
	2.	FIOM Construction Manager.....	165
	a.	<i>Federation Entity Manager</i>	165
	b.	<i>Component Model Correlator</i>	170
	c.	<i>Translation Generator</i>	171
	3.	Translator Information Generator.....	172
	4.	Federation Ontology Manager	174
	5.	FIOM Database	174
	6.	Translation Library	175
	7.	Federation Ontology Database.....	175
	8.	Translator Information Database.....	175

E.	OOMI IDE PROTOTYPE USER INTERFACE DESIGN.....	176
1.	OOMI IDE GUI Components	176
2.	FIOM Construction Phase Folders.....	176
3.	OOMI IDE Toolbar, and Directory and Display Panes	176
a.	<i>ADD Component System External Interface</i>	177
b.	<i>MANAGE Federation Entities</i>	178
c.	<i>REGISTER Component Class Representation</i>	179
d.	<i>UPDATE Federation Ontology</i>	181
e.	<i>GENERATE System-Specific Translator Information</i>	182
F.	SUMMARY.....	183
VI.	COMPONENT SYSTEM OBJECT CORRELATION UNDER THE OBJECT ORIENTED METHOD FOR INTEROPERABILITY (OOMI).....	185
A.	CORRELATION OF COMPONENT SYSTEM AND FEDERATION REPRESENTATIONS OF A REAL-WORLD ENTITY	185
B.	OOMI CORRELATION METHODOLOGY	187
1.	Generating Syntactic and Semantic Information Used in the Correlation Process	188
a.	<i>Generating Components Used By Semantic Matching Process</i>	188
b.	<i>Generating Components Used By Syntactic Matching Process</i>	190
2.	Using Syntactic and Semantic Information to Correlate Component and Federation Representations of Real-World Entities.....	200
a.	<i>Semantic Correlation Process</i>	201
b.	<i>Syntactic Correlation Process</i>	201
C.	SUMMARY.....	205
VII.	OBJECT-ORIENTED METHOD FOR INTEROPERABILITY (OOMI) TRANSLATOR	207
A.	TRANSLATOR OVERVIEW	207
B.	TRANSLATOR ARCHITECTURAL ALTERNATIVES.....	208
C.	TRANSLATOR FUNCTION.....	212
1.	Source To Intermediate Model Translation	212
a.	<i>Converting From XML to Object Representation of Exported Information</i>	213
b.	<i>Translation From Source Object Representation to Intermediate Object Representation</i>	217
c.	<i>Converting From Intermediate Object Representation of Exported Information to XML Instance Document Representation</i>	219
2.	Intermediate To Destination Model Translation.....	221
a.	<i>Converting From XML Document Representation back to Intermediate Object Representation</i>	223
b.	<i>Resolving Differences in View between Received Intermediate Model and Destination Model of Real-World Entity</i>	226

c.	<i>Translation From Intermediate Object Representation to Destination Object Representation</i>	<i>229</i>
d.	<i>Converting From Destination Object Representation to Destination XML Document Representation</i>	<i>230</i>
D.	TRANSLATOR SUMMARY	231
VIII.	CONCLUSION AND RECOMMENDATIONS FOR FUTURE RESEARCH	237
A.	REVIEW OF CRITERIA USED FOR EVALUATING INTEROPERABILITY APPROACHES AND LIMITATIONS SEEN IN CURRENT SYSTEMS	237
B.	EVALUATION OF OBJECT-ORIENTED METHOD FOR INTEROPERABILITY AGAINST INTEROPERABILITY COMPARISON CRITERIA	238
1.	Types of Heterogeneity Addressed	238
2.	Capability for Application of Computer Aid for Model Correlation	240
3.	Required Knowledge of Remote Operations	241
4.	Required Modification to Existing System	242
5.	Translation Methodology	243
6.	Capability for Application of Computer Aid for Translation Development	243
7.	Support for Federation Extensibility	243
8.	Information Exchange Versus Joint Task Execution	244
C.	RECOMMENDATIONS FOR FUTURE RESEARCH	245
1.	Evaluation of Efficiency and Effectiveness of OOMI in Creating an Interoperable Federation of Systems	245
2.	Enhancements to Correlation Methodology	246
a.	<i>Semantic Correlation Methodology</i>	<i>246</i>
b.	<i>Syntactic Correlation Methodology</i>	<i>247</i>
3.	Enabling Join Operations for Federation Entity View Definition	248
4.	Expansion of Correlation Methodology Application During FIOM Construction	248
a.	<i>Application of Correlation Methodology to Mapping of Corresponding Attributes and Operations during Translation Generation</i>	<i>248</i>
b.	<i>Application of Correlation Methodology and Behavioral Equivalence Determination Algorithms to Modification of FIOM to Provide Required One-To-One Correspondence Between CCR and FCR Schemas During CCR Registration</i>	<i>248</i>
5.	Extending OOMI IDE to Operate as a Distributed Network Application	249
6.	Resolution of Modeling Differences in Real-Time Systems	250
D.	CONCLUDING REMARKS	251

APPENDIX A: MODIFYING FIOM TO PROVIDE REQUIRED CORRESPONDENCE BETWEEN CCR AND FCR SCHEMAS DURING CCR REGISTRATION.....	253
A. ADDING NEW FEDERATION ENTITY (FE) TO FEDERATION INTEROPERABILITY OBJECT MODEL (FIOM).....	255
B. ADDING COMPONENT CLASS REPRESENTATION (CCR) TO EXISTING FEDERATION ENTITY (FE)	255
1. Adding CCR to Existing Federation Entity View (FEV).....	256
2. Adding New View to FE.....	257
<i>a. CCR Schema Properties Subset of FCR Schema Properties.....</i>	<i>259</i>
<i>b. FCR Schema Properties Subset of CCR Schema Properties.....</i>	<i>262</i>
<i>c. CCR and FCR Schema Have Properties in Common, But No Subset Relation Exists.....</i>	<i>266</i>
<i>d. No Correspondence Between CCR and FCR Schemas</i>	<i>270</i>
C. SUMMARY.....	275
LIST OF REFERENCES.	277
INITIAL DISTRIBUTION LIST	283

LIST OF FIGURES

Figure I-1.	Quest for System Interoperability	2
Figure I-2.	Impediments to System Interoperability	4
Figure II-1.	OMG's Object Management Architecture (From [Pop98])	19
Figure II-2.	Binary Structure of a COM Object (From [Kol00])	33
Figure II-3.	DCOM Overall Architecture (From [Kol00])	36
Figure II-4.	Use of MIDL Proxy and Stub Constructs for Achieving Programming Language Transparency (From [Kol00])	37
Figure II-5.	Data Types Supported by COM+ (From [Kir97])	38
Figure II-6.	e*Gate Components (From [EGI00])	54
Figure II-7.	e*Gate Architecture (From [EGI00])	56
Figure II-8.	e*Gate Integration Process (From [Smi01])	57
Figure II-9.	e*Index Global Identifier Customer Detail Screen (From [EIGI00])	59
Figure II-10.	Example XML Document	74
Figure II-11.	XML and Java Relationships (From [Rei01])	79
Figure III-1.	Three Elements of Faceted Approach (From [Pri91])	93
Figure III-2.	Multi-level Filtering Model (From [Her97])	109
Figure III-3.	Semantic Integration Procedure (From [LC94, LC00])	119
Figure IV-1.	Object-Oriented Method for Interoperability (OOMI) Key Components	130
Figure IV-2.	Differing Views of Real-World Entity	133
Figure IV-3.	Differing Real-World Entity View Representations	134
Figure IV-4.	OOMI Federation Entity (FE) Archetype	136
Figure IV-5.	Example Views of a Federation Entity with Federation and Component Class Representations	138
Figure IV-6.	OOMI Archetype for Federation and Component Class Representations (FCR and CCR) Showing Constituent Schema, Syntax, and Semantics Classes	139
Figure IV-7.	FCR and CCR Schemas for Example Ground Combat Vehicle	140
Figure IV-8.	FCR-CCR Translation Class	143
Figure IV-9.	FCR Schema Inheritance Hierarchy	145
Figure IV-10.	Translator - FIOM Interaction	149
Figure V-1.	Use-Case Model of Candidate FIOM Construction Process	156
Figure V-2.	OOMI IDE Block Diagram	164
Figure V-3.	OOMI IDE GUI Components	177
Figure V-4.	<i>ADD Component System External Interface Folder</i> Display and Functionality	178
Figure V-5.	<i>MANAGE Federation Entities Folder</i> Display and Functionality	179
Figure V-6.	<i>REGISTER Component Class Representation</i> Display and Functionality	180
Figure V-7.	<i>REGISTER CCR Folder</i> Translation Generation Window	181
Figure V-8.	Translation Generated During <i>REGISTER CCR Phase</i>	182
Figure VI-1.	Back-Propagation Neural Network Architecture in OOMI IDE (After [LC00])	198
Figure VI-2.	Example Neural Network Training Data and Target Result (After [She02])	199

Figure VI-3.	Training OOMI IDE Neural Networks (After [LC00])	200
Figure VI-4.	Discriminator Vectors for Example MechanizedCombatVehicle CCR (After [She02])	202
Figure VI-5.	Using Trained Neural Network to Evaluate Attribute and Operation Correspondence (After [LC00])	203
Figure VI-6.	Example CCR-FCR Comparison Matrix	203
Figure VI-7.	Computing Single Value for CCR-FCR Comparison Matrix	204
Figure VII-1.	Source and Destination System Translator Implementation	208
Figure VII-2.	Source-System-Only Translator Implementation	209
Figure VII-3.	Destination-System-Only Translator Implementation	210
Figure VII-4.	Middleware Translator Implementation	211
Figure VII-5.	Process for Converting Source XML Instance Document to its Equivalent CCR Schema Object	214
Figure VII-6.	Source XML Document “mechanizedCombatVehicle.xml”	215
Figure VII-7.	Source XML Schema “mechanizedCombatVehicle.xsd” Excerpt	216
Figure VII-8.	Converting From XML to Object Representation of Exported Information	218
Figure VII-9.	CCR Schema Object to FCR Schema Object Translation	219
Figure VII-10.	Converting from Source to Intermediate Object Representations	220
Figure VII-11.	Process for Converting FCR Schema Object to its Equivalent XML Instance Document	221
Figure VII-12.	Conversion from Intermediate Object Representation to XML Document Representation	222
Figure VII-13.	Intermediate XML Instance Document “groundCombatVehicle_View1.xml”	223
Figure VII-14.	Process for Converting XML Instance Document to its Equivalent FCR Schema Object	224
Figure VII-15.	Conversion from XML Instance Document Representation back to Intermediate Object Representation	225
Figure VII-16.	Example FCR Schema Inheritance Hierarchy Excerpt	228
Figure VII-17.	FCR Schema Object to CCR Schema Object Translation	229
Figure VII-18.	Translating from Intermediate to Destination Object Representation	230
Figure VII-19.	Process for Converting CCR Schema Object to its Equivalent XML Instance Document	231
Figure VII-20.	Destination XML Schema “armoredFightingVehicle.xsd”	232
Figure VII-21.	Conversion From Destination Object Representation to Destination XML Instance Document Representation	233
Figure VII-22.	Destination XML Instance Document “armoredFightingVehicle.xml”	234
Figure VII-23.	Source to Intermediate Model Translation	235
Figure VII-24.	Intermediate to Destination Model Translation	236
Figure A-1.	No FE in FIOM Corresponding to CCR Being Registered	254
Figure A-2.	New FE Defined With FEV Having Same Perspective of Real-World Entity as CCR Being Registered	256
Figure A-3.	One-To-One Correspondence Between CCR and FCR Schema Properties	257
Figure A-4.	CCR Added to Federation Entity View Exhibiting One-To-One Correspondence Between CCR and FCR Schema Properties	258
Figure A-5.	CCR Schema Properties Subset of FCR Schema Properties	260

Figure A-6.	Generalized FCR Schema Added to FEV	261
Figure A-7.	FEV Containing Generalized FCR Schema Defined for FE.....	262
Figure A-8.	FCR Schema Properties Subset of CCR Schema Properties.....	263
Figure A-9.	Specialized FCR Schema Added to FEV	264
Figure A-10.	FEV Containing Specialized FCR Schema Added to FE	265
Figure A-11.	CCR and FCR Schema Have Properties in Common, But No Subset Relation Exists.....	267
Figure A-12.	Sibling to Existing FCR Schema Added to FEV	268
Figure A-13.	FEV With FCR Schema Sibling to Existing FCR Schema Added to FE	269
Figure A-14.	No Correspondence Between CCR Schema Being Registered and Existing FCR Schemas in FE	272
Figure A-15.	New Root FCR Schema With Child FCR Schema Corresponding to CCR Schema Being Registered Included with FEV	273
Figure A-16.	FEV With FCR Schema Sibling to Previous Existing Root FCR Schema Added to FE	274

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table II-1.	Evaluation of CORBA Support for Resolution of Modeling Differences	24
Table II-2.	Evaluation of COM, DCOM, and COM+ Support for Resolution of Modeling Differences.....	39
Table II-3.	Evaluation of J2EE Support for Resolution of Modeling Differences.....	49
Table II-4.	Evaluation of SeeBeyond Support for Resolution of Modeling Differences..	60
Table II-5.	Evaluation of HLA Support for Resolution of Modeling Differences.....	69
Table II-6.	Evaluation of XML Support for Resolution of Modeling Differences	80
Table III-1.	Metadata and Data-Content-Based Discriminators Used in SEMINT (From [LC00]).....	121
Table IV-1.	Comparison of Multidatabase Heterogeneity Resolution Methods (after [HL96]).....	128
Table VI-1.	XML Schema Fields Used for Keyword Determination (From [Pug01])	189
Table VI-2.	Metadata Based Discriminators Used in Syntactic Correlation Process (After [She02])	193
Table VI-3.	Discriminator Values Used for Syntactic Correlation (After [She02]).....	196
Table VI-4.	Discriminator Values Used for Syntactic Correlation (continued) (After [She02])	197
Table VIII-1.	Evaluation of OOMI Support for Resolution of Modeling Differences	239

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to offer my sincere thanks to my dissertation supervisor, Dr. Luqi, and to all of the members of my dissertation committee: Dr. Valdis Berzins, Dr. Edmund Freeman, Dr. Jun Ge, Dr. William Kemple, and Mr. Richard Riehle. Your guidance, support, and encouragement have enabled me to better understand the complexities involved in achieving system interoperability. This understanding has permitted me to make significant contributions toward resolving modeling differences among systems, a key factor for enabling the sharing of information and tasks among heterogeneous systems.

Although not members of my committee, Dr. Mantak Shing, Dr. Craig Rasmussen, and Dr. Bret Michael also supported me in my studies. They were always willing to take the time to help, regardless of their many other responsibilities.

I would also like to thank the members of the interoperability working group who have helped and assisted me along the way: LT Todd Ehrhardt, CPT Bryan Lyttle, Capt Randy Pugh, MAJ Brent Christie, Mr. Shong Cheng Lee, LT Steve Shedd, and LT George Lawler. With your help, I was able to turn my abstract, often vague notion of a federation interoperability model into a concrete, demonstrable foundation for continued research.

Thanks to SPAWAR Systems Center San Diego for providing funding support for this research under their SPAWAR Fellowship program. Equipment and travel funded by this fellowship were key enablers of this effort.

And finally, a special thank you to my wife, Betty, and daughters, Elizabeth and Emilee. You have done so much to enable me to focus on this sometimes seemingly insurmountable challenge. Without your support, this would not have been possible!

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. QUEST FOR SYSTEM INTEROPERABILITY

Past acquisition and development practices in the Department of Defense (DoD) have led to the procurement of numerous special-purpose, non-interconnected software-intensive systems for application areas varying from embedded weapon system software to logistic management systems. Advances in computer communications technology, the recognition of common areas of functionality in related systems, and an increased awareness of how enhanced information access can lead to improved capability are driving an interest toward integration of current stand-alone systems to meet future system requirements. In addition, the integration of Commercial Off-the-Shelf Software (COTS) and Government Off-the-Shelf Software (GOTS) with existing legacy systems offers an attractive alternative for enhancing the capabilities of these systems without incurring the expense and time required for a new software development.

An example of where independently developed systems might be interconnected to provide an increase in capability over that provided by the individual components is illustrated in Figure I-1. In the figure, a ground-based forward observer with a hand-held Battlefield Digital Assistant (BDA) gathers intelligence and targeting information about the field of battle. The information gathered by the forward observer is transmitted via a battlefield wireless network to an in-theater intelligence cell where it is processed on a Command and Control, Computers, Communication, and Intelligence (C⁴I) system for forwarding to a sea-based task force. The information is relayed to a strike planning team onboard the task force aircraft carrier where the targeting information is used to plan a Tomahawk strike mission using the Tomahawk Planning System (TPS) [TWS02]. Strike mission data is forwarded to a task force launch platform where the planned mission is loaded into the Advanced Tactical Weapons Control System (ATWCS) used to launch a Tomahawk strike [TWS02]. The launch platform then executes the mission to destroy the target initially designated by the forward observer. In the scenario portrayed above, a number of independently developed systems are interconnected to provide a system-of-systems whose combined capability exceeds that of the individual components.

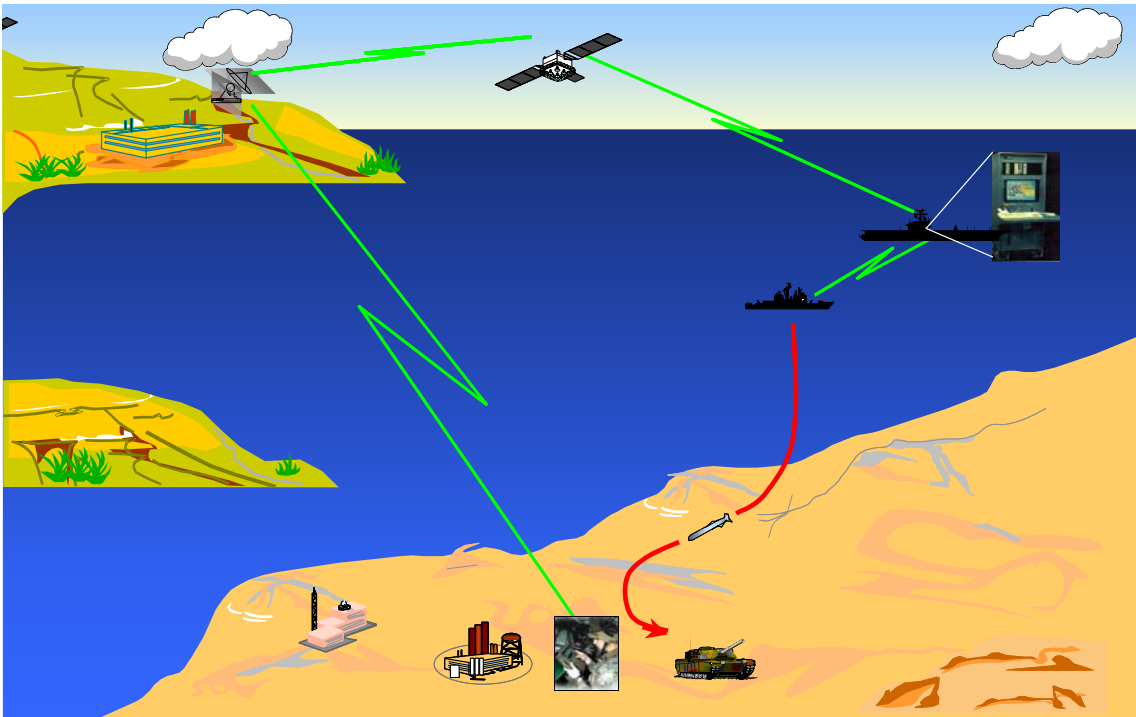


Figure I-1. Quest for System Interoperability

By interconnectivity, I refer to the ability of systems to communicate and exchange information. Merely interconnecting these systems is not sufficient for achieving the capability improvement desired. Full realization of the potential synergistic benefits can only be achieved if system interoperability is attained. System interoperability involves not only the ability of systems to exchange information but also includes the capability for interaction and joint execution of tasks. [LISI98, Pit97]. Therefore, the objective in creating such a system-of-systems is not merely providing system interconnectivity, but in achieving an interoperable *system federation*.

In this dissertation I make a distinction between a federation of systems and an integrated system of components. I use the term *integrated system* to describe an interconnected compilation of homogenous components produced by a development team that shares common objectives and has a common view of the problem environment being modeled. The term *system federation* is used to describe an interconnected collection of independently developed heterogeneous systems or components. The method presented in this dissertation for achieving system interoperability focuses on the

system federation- how to achieve interoperability among a number of independently developed systems or components that were not originally intended to interoperate. If constructed properly, component interoperation should be one of the key design requirements for the integrated system and the types of heterogeneities for which the proposed method is designed to resolve should not be a factor.

A prime difficulty in achieving interoperability among heterogeneous components of a system federation is that the component systems were developed independently, without any requirement for interaction. Thus systems may have different architectures, different hardware platforms, different operating systems, different host languages and different data models. For example, as indicated in Figure I-2, each component system might have a different model of the vehicle being targeted by the forward observer. For instance, the forward observer's BDA might be implemented on a Palm wireless handheld device, running Palm-OS and a special-purpose targeting application. The data model used to represent the targeted vehicle employs a *MechanizedCombatVehicle* record structure with elements *mcvType* used to indicate whether the vehicle is a tank, personnel carrier, or reconnaissance vehicle; *mcvLocation* providing the vehicle's location using Military Grid Reference System (MGRS) coordinates; *mcvTime* giving the time of observation at the specified location in Local Mean Time (LMT); and *mcvRadius* specifying the maneuvering range of the vehicle in kilometers (km).

Similarly, the C⁴I system might be implemented on a Microsoft Windows™ based workstation implemented in C++. The C⁴I system represents the targeted vehicle using an *ArmoredMilitaryVehicle* structure containing elements *designation* specifying the type of vehicle (main battle tank, missile launcher, armored personnel carrier, etc.); *position* providing the vehicle's coordinates using latitude and longitude; and *time* in Greenwich Mean Time (GMT) providing the moment when vehicle observation was made and its position recorded.

Finally, the unix-based TPS is implemented in Ada and uses an *ArmoredFightingVehicle* record structure to model the targeted vehicle. The *ArmoredFightingVehicle* record includes components *afvClassification* specifying the type of vehicle (battle tank, rocket launcher, truck, etc.); *afvLocation* providing the vehicle coordinates using latitude

and longitude; *afvObsTime* giving the time of observation of the vehicle at the specified location in GMT; and *afvStatus* indicating whether the vehicle is operational, damaged, or destroyed.

Short of redeveloping a new system using the consolidated requirements from the various component systems and a common architecture, hardware platform, operating system, host language, etc. (a cost prohibitive approach), a means must be devised to achieve the goal of component interoperability in the face of expected limited acquisition budgets.

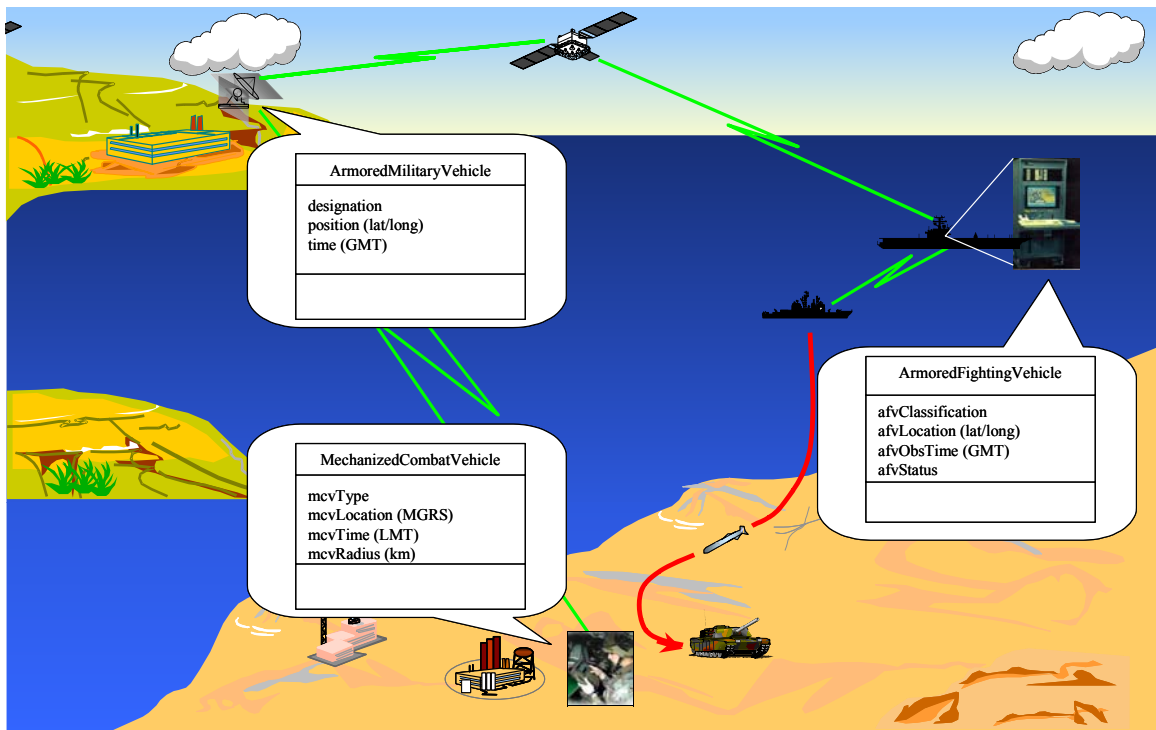


Figure I-2. Impediments to System Interoperability

B. LIMITATIONS WITH CURRENT APPROACHES TOWARD ACHIEVING INTEROPERABILITY

Current approaches to achieving interoperability among heterogeneous systems include several limitations. First, they do not provide a means for resolving the complete spectrum of modeling differences found among heterogeneous systems. Second, they do not provide assistance in determining when different system models refer to the same entity from the problem domain. Third, in order to access another component or system's state or exercise its behavior, most current approaches require the requesting system to

utilize the provider system's model of its state or behavior to access its information. This usually requires modification to the existing systems in order to enable interoperation, significantly limiting the applicability of the approach when constructing a system federation from existing components where component modification is restricted by cost or other concerns. Fourth, most approaches utilize a direct point-to-point conversion process for resolving modeling difference among systems vice a two-step conversion process using an intermediate model. For a federation of more than three systems, a point-to-point approach requires a greater number of translations to be defined than does the two-step process. Fifth, most approaches provide no or limited support to development of the translations required to resolve modeling differences among systems. Finally, most approaches are concerned only with the resolution of modeling differences for information exchanged among systems and do not provide the capability for resolving possible differences in the signatures used to access the behavior of corresponding methods on different systems. One of the underlying causes of the limitations found in current approaches toward achieving interoperability is the failure to provide a comprehensive model of the information exported from or imported to systems in a federation. Capture of this information is critical in order to be able to identify opportunities for information exchange and joint task execution and to identify areas where modeling differences among systems must be resolved.

C. RESEARCH QUESTION AND HYPOTHESIS

To overcome the limitations identified in Section I.B above, I explore technologies and methods to provide an answer to the following question: Given N heterogeneous systems, can we resolve the differences in data models and ensure consistency in data mapping to enable interoperability among the systems? In response to the above question I offer the following hypothesis: By using a model-based approach, a computer-aided methodology can be provided to aid in the resolution of data modeling differences among systems targeted for integration in order to enable system interoperability. The Object-Oriented Method for Interoperability (OOMI) presented in this dissertation provides such a methodology.

D. OBJECT-ORIENTED METHOD FOR INTEROPERABILITY (OOMI) OVERVIEW

The current state-of-the-art for integration of heterogeneous systems involves manually resolving differences in data modeling and mapping for each interface between systems, in an inherently customized manner. The first step in advancing the state-of-the-art is to develop a general model of the interoperation among systems. This model, termed a Federation Interoperability Object Model (FIOM) under the OOMI, captures the various component system models of the state and behavior information shared between systems as presented by a component system's external interface. The FIOM also captures syntactic and semantic information about a component model in order to enable an *interoperability engineer* to determine whether two component models refer to the same entity in the problem domain. By interoperability engineer I refer to the person or persons charged with FIOM construction. Expected qualifications for an interoperability engineer are provided in Section V.B and their role in the FIOM construction process detailed in Chapter V and Appendix A. Finally, the FIOM includes the mechanisms required for resolving differences among component models.

While creation of a model to capture the interoperation among systems is itself an advancement over the methods currently used for system interoperability, the true benefits of the OOMI lie in the foundation provided by the FIOM for application of computer aid. The first application of computer aid comes in the construction of an FIOM for a specified federation of systems. The OOMI includes an Integrated Development Environment (IDE) to aid the interoperability engineer in FIOM construction during federation development prior to runtime. The IDE provides computer aid to the interoperability engineer in the areas of component model correlation, translation definition, and FIOM construction.

The final area where computer aid is applied toward achieving system interoperability under the OOMI is in the runtime resolution of modeling differences among systems in a federation. The OOMI includes *translators* that use information contained in the FIOM created prior to runtime for resolving heterogeneities among federation systems at runtime.

E. CONTRIBUTIONS PROVIDED BY THIS DISSERTATION

Development of a model for capturing the shared state and behavior of the components of a system federation, the use of computer aid in model construction, and use of the resultant model to automatically resolve heterogeneities in the state and behavior information shared among federation components are the principal contributions of the OOMI presented in this dissertation. Furthermore, the OOMI provides a method for addressing the resolution of the complete spectrum of heterogeneities found among systems, something the previous interoperability work does not accomplish. Key elements of these contributions include:

- Providing a model to capture state and behavior shared among systems in a federation; extremely valuable aid in determining what state and behavior information is available for sharing among systems, particularly when modifications are required to an existing federation developed under the OOMI;
- Classification of modeling differences among autonomously developed, heterogeneous systems as differences in *what* is being modeled (view) and differences in *how* the modeled information is represented (representation);
- Introduction of a *view inheritance hierarchy* to capture the relationships among different system models, and the exploitation of Liskov and Wing's behavioral notion of subtyping to determine when one model of a real-world entity's state and behavior may be suitable for use by another [LW94, WO00];
- Introduction of a set of interoperability engineer defined translations to resolve differences in representation between models having the same view of a real-world entity;
- Application of computer-aid for establishing correspondence among different models of information shared among systems;
- Use of computer-aid in creating the translations used to resolve modeling differences among systems;
- Automation of the run-time process used for resolving modeling differences among the state and behavior shared between systems;
- Ability for achieving interoperability among a federation of legacy systems without requiring extensive modifications to the existing systems.

F. IMPACT AND LONG-TERM SIGNIFICANCE OF DISSERTATION'S CONTRIBUTIONS

Integration of heterogeneous legacy systems has historically been an essentially manual, labor intensive, and costly evolution. The ability to automate part or all of this integration process holds the promise of providing enhanced capability at significant time and cost savings.

While the focus of this dissertation is to provide a means for enabling existing systems to interoperate, applicability of the OOMI is not limited to this context. The goal for new system development should be to include requirements for interoperability into the system design. However, the relevance of the OOMI will not be lost even if interoperability requirements are designed into systems from the start. The desire to add new capability to an existing system without redevelopment will make the contributions provided in this dissertation relevant even if the goal of including interoperability requirements into a system's design is achieved. In addition, the same methodology for integrating heterogeneous legacy systems can be applied to the integration of COTS and GOTS components with existing systems to enhance their capability while minimizing cost, an attractive possibility for any area of potential application.

G. DISSERTATION ORGANIZATION

The remainder of this dissertation is organized as follows. In Chapter II a survey of previous work toward achieving interoperability among independently developed systems is conducted. In this chapter I look at the causes and types of modeling differences among systems and provide a comparison of the most pertinent existing approaches to attaining interoperability among heterogeneous systems using a set of common criteria for system evaluation.

Chapter III provides background information and theory on potential methods to be used for identifying correspondences among information exported or imported by different systems in a federation. Identification of correspondences among information is essential for determining whether systems can interoperate. Unless there are commonalities in the information modeled by two systems, interoperation is not possible. Included in this chapter is an identification of a number of measures for evaluating the effectiveness of various correlation methodologies as well as a discussion of several methodologies considered for implementation in the OOMI IDE.

In Chapter IV I introduce the Object-Oriented Method for Interoperability (OOMI) and provide an overview of the method's major components. The Federation Interoperability Object Model (FIOM) captures the state and behavior shared among systems in a federation as well as the information needed to establish correspondences among component models and to resolve any modeling differences between

corresponding systems. The OOMI Integrated Development Environment (IDE) is used for constructing an instance of the FIOM for a specified federation of systems. The translator uses information captured in a specified FIOM to resolve modeling differences among federation components at runtime.

Chapter V discusses the OOMI IDE purpose and identifies a number of considerations that should be taken into account in constructing an IDE for use in FIOM development. A process for FIOM development is also introduced from which opportunities for application of computer aid are identified. Finally, an initial prototype OOMI IDE is presented and an overview of its major components and Graphical User Interface (GUI) provided.

Chapter VI discusses the opportunities for correlation method application during FIOM construction. The design of the correlation methodology chosen for the OOMI IDE implementation is also covered.

Chapter VII covers the translator used under the OOMI to resolve modeling differences among federation components during their runtime operation. Included in the discussion is a look at the architectural alternatives for translator implementation as well as a discussion on translator functionality used when converting exported state and behavior information from the source model to the equivalent model to be used for destination system importation.

Chapter VIII revisits the research question presented in Section I.C to evaluate the OOMI's success in satisfying the research hypothesis. From this evaluation a conclusion is reached and areas for future research suggested.

THIS PAGE INTENTIONALLY LEFT BLANK

II. SURVEY OF PREVIOUS WORK ON ACHIEVING INTEROPERABILITY AMONG INDEPENDENTLY DEVELOPED SYSTEMS

A. MODELING DIFFERENCES AMONG SYSTEMS

1. Causes of Modeling Differences

Chapter I provided an example of how modeling differences among systems can impact their interoperability. Before looking at the types of modeling differences that can arise in independently developed systems, it is important to look at why these differences occur in the first place. In their research related to database schema integration, Batini et al. [BLN86] described three major causes of representational heterogeneity:

Different perspectives. The different needs of users, program managers, and design teams can lead to differences in data representations even when modeling the same information.

Equivalent constructs. Equivalent models of the same real-world domain can be created using different combinations of the same basic modeling constructs.

Incompatible design specifications. Different application design specifications can result in different database schemas for the same real-world domain.

While originally cited in the context of database schema integration, these factors also apply directly to the types of model heterogeneity found in autonomously developed systems.

2. Kinds of Modeling Differences

Early work in multidatabase architectures categorized modeling differences found in heterogeneous database systems. One of the pioneers in early multidatabase efforts, Gio Wiederhold, defined seven classes of heterogeneity found in autonomously developed database systems [Wie93]. The kinds of heterogeneity defined for databases closely relate to the kinds of heterogeneity found in the interoperability context. Using Wiederhold's classification as a baseline and reflecting views from Hammer and McLeod [HM99], Holowczak and Li [HL96], Kim and Seo [KS91], and Kahng and McLeod [KM98], I describe eight classes of heterogeneity found when trying to achieve interoperability among a federation of independently developed systems. I have added *heterogeneity of structure* to Wiederhold's list and renamed *heterogeneity of*

representation to *heterogeneity of presentation* to eliminate potential overloading in the use of the term *representation*. As a result, the following classification of modeling differences is used in this document:

- Heterogeneity of Hardware and Operating Systems
- Heterogeneity of Organizational Models
- Heterogeneity of Structure
- Heterogeneity of Presentation
- Heterogeneity of Meaning
- Heterogeneity of Scope
- Heterogeneity of Level of Abstraction
- Heterogeneity of Temporal Validity

a. Heterogeneity of Hardware and Operating Systems

Heterogeneity of hardware and operating systems relates to differences in the hardware and operating system platforms encountered when integrating autonomously developed systems [HM99, Wie93]. The continual evolution of computer hardware almost guarantees differences in the platforms used to host the components of a system federation, particularly if the federation is constructed or modified over a period of time. Hardware platform differences can result in differences in the physical format of information on two systems, such as the word size used to represent the primitive language types, or style of data format (Big Endian versus Little Endian) [Fei99]. Although variety in operating system types appears to be converging to three major families- Microsoft Windows®, Apple Macintosh, or UNIX derivatives- evolution within these families will ensure that we must continue to contend with operating system differences.

b. Heterogeneity of Organizational Models

Heterogeneity of organizational models refers to differences in the conceptual models used by autonomously developed systems. In the context of multidatabase systems, conceptual model differences relate to dissimilarities in the database models used, such as network, hierarchical, relational, universal, or object structured [HM99, Wie93]. In the context of interoperability, heterogeneity of organizational models can refer to differences in analysis and design principles

employed, such as use of an *Object-Oriented Analysis and Design (OOAD)* approach versus a *structured analysis* approach [Pre01].

c. *Heterogeneity of Structure*

Variations in the structure of how information is arranged can occur among systems using the same organizational model. These variations can include differences in structural composition, possible schema mismatches, and variations due to the presence of implied information. Differences in structural composition arise when a real-world entity is modeled as an object on one system and an attribute in another, e.g., such as an *aircraft route* being modeled as an attribute of an *aircraft mission* object on one system (as one element of the overall mission) and as a separate entity used for deconflicting missions in another [HL96]. Schema mismatches can occur when similar concepts are modeled differently in the schemas of corresponding systems, e.g., a relationship that is modeled as one-to-one in one schema and one-to-many in another [HM99]. Finally, structural differences can arise when information that is explicitly modeled in one system is implicitly defined in another. An example of this kind of structural heterogeneity is seen when the type of an object is explicitly specified in one schema and implied from the object's name in another [KS91].

d. *Heterogeneity of Presentation*¹

Heterogeneity of presentation includes domain mismatch problems, the use of different units of measure, differences in precision, disparate data types, and different field lengths or variations in integrity constraints. Domain mismatch problems occur when the same concept is characterized differently in two separate systems, such as geographic position measured in latitude and longitude on one system and Military Grid Reference System (MGRS) on another [HM99]. Systems may also use different units of measure when quantifying the same object, i.e., yards in one system versus meters in another for distance measurement. Differences in precision may also occur between systems, i.e., one system might measure range to a target in hundreds of yards (more precise) versus another system measuring the same quantity in thousands of yards (less

¹ The term presentation is taken from computer networking, where the OSI presentation layer defines the format of the data to be exchanged between applications and provides transformations between data formats [Sta00].

precise) [HM99]. Systems may express the same characteristic using disparate data types: one system represents a telephone number as an integer while another characterizes it as a character string. Systems may also use different field lengths for defining the same entity [Wie93]: one system may provide a twenty-character description of a weapon's capability while another only allows ten characters for the same information. Similarly, two systems might place different integrity constraints on the same value [KS91]. For instance, one system might allow a value for missile effectiveness in the range of one to twenty nautical miles whereas a different system might allow effective range values up to thirty nautical miles.

e. Heterogeneity of Meaning

Heterogeneity of meaning results from the imprecise nature of natural language for characterizing a real-world entity. The use of *homonyms*, *synonyms*, and *abbreviations* for specifying real-world entity features contribute to this type of difference. Homonyms refer to the use of the same word to convey different meanings, such as the use of the word *tank* to describe both a tracked combat vehicle and a container for liquid or gaseous material storage [HL96, KM98, Wie93]. Synonyms refer to the use of different words to describe the same real-world entity or characteristic. For example, both *location* and *position* can be used to refer to the geographic coordinates of a military unit. The use of abbreviations for depicting real-world entities represents a special case of the use of synonyms where different abbreviations can be used to represent the same entity, such as the use of *POSIT*, *PSIT*, or *POS* to refer to the position of an entity [HM99, KS91].

f. Heterogeneity of Scope

Heterogeneity of Scope results from differences in the information used to model a real-world entity. These differences can arise from different perspectives on what attributes a given application needs to capture about the real-world entity being modeled [Wie93, HL96]. For example, a logistics management system might include attributes *fuelCapacity* and *ammunitionStatus* for a main battle tank, whereas a command and control system would include attributes *weaponRange* and *defensiveArmor* in its tank model.

g. Heterogeneity of Level of Abstraction

Heterogeneity of level of abstraction results from differences in the level and degree of aggregation of atomic data elements. For example, one system may store sales information as a monthly total while another system aggregates the same basic data as a yearly sum [Wie93, HL96].

h. Heterogeneity of Temporal Validity

Finally, modeling differences may arise from differences in the time used by two models to observe or record the state of a real-world entity. A difference in the length of time data remains valid is another source of heterogeneity of temporal validity. For example, two companies may record their accounting information according to different fiscal years. These differences in temporal validity are particularly an issue with military C⁴I systems [HL96, Wie93].

B. CRITERIA FOR EVALUATING INTEROPERABILITY APPROACHES

The first criterion for evaluating alternative interoperability approaches is a determination of the types of heterogeneity that may be resolved using the approach. In addition, I have selected seven other criteria for comparing existing interoperability approaches. These criteria are listed below and discussed in the following paragraphs.

- Types of heterogeneity addressed
- Capability for application of computer aid for model correlation
- Required knowledge of remote operations
- Required modification to existing system
- Translation methodology
- Capability for application of computer aid for translation development
- Support for federation extensibility
- Information exchange versus joint task execution

1. Types of Heterogeneity Addressed

Section II.A.2 suggested a classification of the modeling differences that must be resolved between systems in order for them to interoperate. Evaluations of the types of heterogeneity addressed as well as the degree to which they are successful at resolving these differences are central to any evaluation of candidate interoperability approaches. For each candidate interoperability approach, an assessment is provided indicating

whether the approach provides a method for resolving each of the eight types of heterogeneity completely, partially, or not at all.

2. Capability for Application of Computer Aid for Model Correlation

Creating a federation of autonomously developed heterogeneous systems involves many real-world entities and potentially numerous models of those entities by the various systems. Manual correlation of different models of each real-world entity could be a difficult, time-consuming chore. In integrating database systems, the data correlation problem poses one of the biggest challenges, with the time required to match corresponding elements between databases requiring as much as four hours per element when done manually [LC00]. The correlation problem is no less formidable in the context of attempting to find correspondences among attributes and operations from different systems' models. Therefore, as the number of systems being integrated, and correspondingly, the number of modeled entities increases, application of computer aid to the correlation process is warranted. An evaluation of the candidate interoperability approaches is made indicating whether they provide assistance for correlating different models of the real-world entities involved in system interoperation. Where provided, the approach is evaluated to determine the extent of assistance given.

3. Required Knowledge of Remote Operations

The ability for one system to invoke operations implemented on another system is one of the tenets for declaring that the two systems are interoperable. In some interoperability approaches a designer must have prior knowledge of the operations available on a remote system (and possibly modify the calling system to comply with a server's interface) in order to take advantage of their functionality. Other approaches enable the designer to invoke a server's methods using the client's representation for the method name and parameters. Such late binding enables independently developed systems to take advantage of operations not known at development time. The candidate interoperability approaches are evaluated to determine whether they enable remote method calls using a client's representation for a method's name and parameters or whether they require the system designer to satisfy the representation specified by the server's interface for these parameters.

4. Required Modification to Existing System

A significant impetus for addressing system interoperability comes from attempts to interconnect independently developed systems that were never intended to interoperate. Typically such existing systems were developed without any of the constructs normally included when forward-fitting a system to support interoperability. Any modification to existing systems to enable them to interoperate is costly and time-consuming. Therefore, methodologies that will enable systems to interoperate without requiring modification to existing software are highly desirable. An evaluation of the candidate interoperability approaches is made to determine if the approach can be applied to existing systems without requiring their modification or if approach compliance must be included in initial system design and development.

5. Translation Methodology

Early interoperability attempts involved the creation of custom point-to-point interfaces between systems. This pair-wise approach to resolving representational differences between systems potentially requires $n(n-1)$ translations for a federation of n systems. Employing a platform-independent intermediate representation during a two-step conversion requires $2(n)$ translations, which is a significant improvement over the pair-wise approach when n is greater than 3. Candidate approaches are evaluated to determine if translations are defined in terms of a direct source-to-destination conversion or whether a two-step process utilizing an intermediate representation is employed.

6. Capability for Application of Computer Aid for Translation Development

Much of the work in creating translations for resolving modeling differences is repetitive, error prone, and can benefit from computer aid. Two opportunities for computer aid that immediately come to mind are: 1) assistance in generating translations between corresponding models, and 2) assistance in reusing commonly used translations. Considering the mundane and detailed nature of such translations, computer aid is warranted, particularly when integrating a large number of systems. An evaluation is made of candidate approaches to determine their capabilities for providing computer assistance in these or other areas of translation definition.

7. Support for Federation Extensibility

Extensibility of a programming language refers to the capability to enrich the language by adding new features or modifying existing ones [RRH00]. A program or design is considered extensible if enhancements can be made to an existing component or data structure without adversely impacting components or applications dependent on the original entity. Similarly, a federation of interoperable systems is considered extensible if additional systems can be added to the federation and changes can be made to the information and operations exchanged among systems without adversely affecting interoperation of the original system federation. An approach that supports construction of an extensible system federation would enable the federation to be implemented in an incremental fashion, using a previously defined version as a baseline for extension vice creating a new federation each time the composition of information or operations exchanged among systems changes. Such extensibility provides the foundation for federation reuse, enabling previously defined artifacts to be reused in future interoperability contexts.

An interoperability approach that supports construction of an extensible system federation is highly desirable in order to enable incremental development and reuse of federation artifacts. Candidate interoperability approaches are compared to determine the level of support provided for creating an extensible system federation. An approach is considered to provide full support for federation extensibility if it enables both the addition of new systems to the federation and modification to existing information and operations shared among systems without impacting the interoperation of the original systems in the federation. Candidate approaches satisfying only one of the above criteria are considered to provide partial extensibility support while approaches satisfying neither of these criteria are considered to provide no extensibility support.

8. Information Exchange Versus Joint Task Execution

Pitoura defines interoperability as the capability of systems to exchange information and to jointly execute tasks [Pit97]. Full interoperability allows systems to take advantage of functionalities and services that would otherwise not be available or would have to be implemented. Candidate approaches are evaluated to determine whether they provide the capability for resolving system heterogeneities only during

information exchange or if their methodologies can also be applied for achieving joint task execution among systems.

C. APPROACHES FOR ACHIEVING INTEROPERABILITY AMONG HETEROGENEOUS SYSTEMS

1. Common Object Request Broker Architecture (CORBA)

a. CORBA Overview

The Object Management Group (OMG) is “an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications” [OMG01]. The OMG Charter includes the requirement to “provide a common architectural framework for object-oriented applications based on widely available interface specifications [Ros98, p.12].” The OMG achieves its goals in this area with the establishment of the Object Management Architecture (OMA) of which the Common Object Request Broker Architecture (CORBA) is a part. The OMA is a set of standards that provides a common architectural framework on which applications are built. The OMA, depicted in Figure II-1, consists of:

- An Object Request Broker (ORB) function
- Object services (known as CORBA services)
- Common facilities (known as CORBA facilities)
- Domain interfaces
- Application objects [Pop98]

CORBA serves to implement the ORB function specified as part of the OMA.

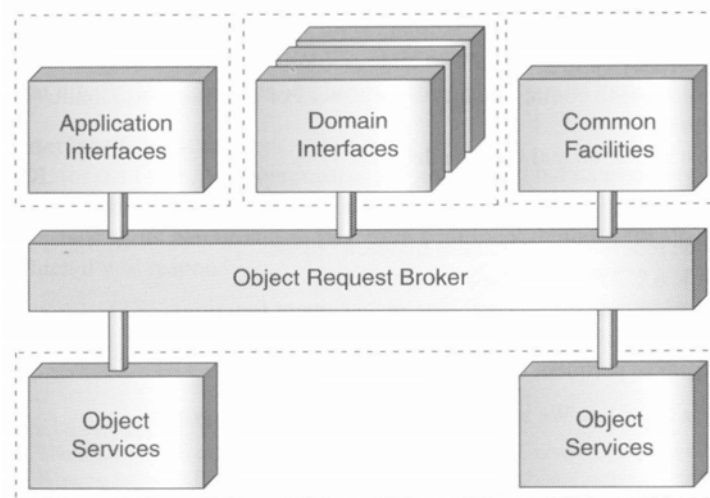


Figure II-1. OMG's Object Management Architecture (From [Pop98])

CORBA provides capabilities in three areas to support interoperability: 1) it provides a standard mechanism for defining the interfaces between components; 2) it specifies a number of standard services such as directory and naming services, persistent object services, and transaction services that are available to all CORBA compliant applications; and 3) it provides the mechanisms to allow application components or separate applications to communicate with each other. These capabilities are provided in a platform independent and language independent fashion.

- Platform independence: CORBA objects can be used on any platform for which there is a CORBA ORB implementation.
- Language independence: CORBA objects can be implemented in a number of programming languages; CORBA objects don't need to know in which language other CORBA objects are implemented in order to be able to communicate. [Ros98]

The initial CORBA standard was released in 1990 as CORBA 1.0. The CORBA standard continues to evolve, with the latest version, CORBA 2.6, released in December 2001 [CORBA01].

b. CORBA Architecture Overview

As it's name implies, CORBA is an object-oriented architecture. This means that CORBA utilizes features from the Object-Oriented Analysis and Design (OOAD) paradigm, such as interface inheritance and polymorphism, to achieve the goals specified in the OMA. This does not mean that CORBA is limited for use with object-oriented systems. In fact, the CORBA architecture provides components required for achieving interoperability among both object-oriented and non-object-oriented systems. The CORBA architecture utilizes the following components in achieving this goal: 1) an Object Request Broker (ORB), 2) an Interface Definition Language (IDL), 3) the CORBA Communications Model, 4) the CORBA Object Model, 5) Clients and Servers, and 6) CORBA services and CORBA facilities.

(1) Object Request Broker (ORB). An ORB is a software component whose purpose is to facilitate communications between objects. When an application component wants to use a service provided by another component, it must first obtain a reference for the object providing that service. The ORB provides this function by resolving requests for object references, thereby enabling application

components to establish connectivity with each other. As part of this capability the ORB provides the functionality for locating a remote object, given an object reference, and for the marshaling of parameters and return values to and from remote method invocations. The CORBA ORB defines the standard for implementing the OMA ORB capability.

(2) Interface Definition Language (IDL). One of the key contributors toward achieving interoperability between both object-oriented and non-object-oriented components is the OMG's Interface Definition Language (IDL). IDL is used to specify the interface between CORBA objects. IDL consists of both a language for specifying an object's interface as well as a translator for mapping the interface specification to the language and system-specific implementation of that interface. As implied, IDL is used to define the interface only; implementation of the interface is done in some other language. Because interfaces defined in IDL can be mapped to virtually any programming language and IDL serves as the common vernacular that all applications and components understand, CORBA can be utilized to connect applications and components implemented in a variety of languages.

(3) CORBA Communications Model. In the CORBA Communications Model, communication between objects takes place between a client and a server. A client is an application that uses the services of a CORBA object, i.e. an application that invokes a method or methods on other objects. Conversely, a server is an application that creates CORBA objects and makes the services provided by those objects available to other applications. The CORBA Communications Model is based on the use of object references (more precisely Interoperable Object References (IORs)) for identifying objects whose services another component might require. When a component of an application wants to access a CORBA object it must first obtain an IOR for that object. Once an IOR is obtained for the object providing the desired service, actual communications between a client and a server are accomplished using a General Inter-ORB Protocol (GIOP) compliant protocol. GIOP specifies a standard for communication between various CORBA ORBs and components. GIOP provides a general specification for inter-ORB communication; specific protocols such as the Internet Inter-ORB Protocol

(IIOP) for TCP/IP networks provide network specific implementations of the GIOP specification.

(4) CORBA Object Model. In addition to the communications model that specifies how communications between objects occur, CORBA features an object model that describes how objects are represented in the system. Because CORBA was designed for distributed systems use, its object model possesses several characteristics tailored for this domain. The first characteristic of CORBA's object model is its semitransparent support for object distribution. Thanks to the use of client stubs, a remote method call looks exactly like a local method call. Thus, object location is transparent to the application that invokes one of the object's methods. A second characteristic of CORBA's object model is that visibility to objects is provided only through passing of references to those objects vice passing the object by value. This methodology grants visibility of an object to another process while retaining ownership of that object by the process in which it is defined. As a result, execution of the object methods takes place within the memory and process space of the owning process.

As previously mentioned, communication between objects is accomplished through the use of an ORB. In achieving its objectives the ORB provides a number of functions ranging from user authentication, to object activation, to object persistence. Access to these functions is provided to a CORBA object by means of a Basic Object Adapter (BOA). Thus, the BOA provides a common set of methods by which an application can access ORB functionality to accomplish inter-object communication.

(5) CORBA Clients and Servers. As mentioned previously, communication between objects in CORBA takes place between a client and a server. To facilitate this communication and to achieve interoperability between clients and servers implemented in a variety of programming languages, CORBA presents the concept of *client stubs* and *server skeletons* to connect a language independent IDL interface specification to the language-specific code that implements the interface. A client stub is a small piece of code that allows a client component to access a server component.

Correspondingly, a server skeleton is a piece of code that allows a server to accept access requests from a client.

The client stub provides a dummy implementation for each method in the interface, thus making a particular CORBA server interface available to the client. The client stub methods are used to marshal and unmarshal parameters for communication with the ORB. On the other side of the interoperation, the server skeleton provides the framework on which the implementation code for a particular interface is built. For each method of an interface, the IDL compiler generates an empty method in the server skeleton with the developer providing the implementation for the server skeleton methods.

(6) CORBAServices and CORBAFacilities. CORBAServices and CORBAFacilities provide a set of standardized capabilities for use by all applications. These capabilities include event management, licensing, object persistence, naming, security, transactions, user interface management, and data interchange, etc. These capabilities supplement the basic use of IDL for creating component interfaces for a specified application service, and then for developing clients to exploit the provided services. In maintaining consistency with the overall CORBA architecture, interface to these services and facilities is specified using IDL. Implementation of the CORBAServices and CORBAFacilities interfaces are vendor dependent; all products may not include a full implementation of these capabilities. [Pop98, Ros98]

c. Evaluation of Interoperability Approach

In this section, I evaluate CORBA against the factors defined in Section II.B for comparing alternative approaches for achieving interoperability among heterogeneous systems. The results of this comparison are summarized in Table II-1 and discussed as follows.

Table II-1. Evaluation of CORBA Support for Resolution of Modeling Differences

Evaluation Criteria	CORBA
Types of Heterogeneity Addressed	Hardware and Operating System; Organizational Models; Presentation (Partial)
Capability for Application of Computer-Aid for Model Correlation?	Partial. CORBA Naming and Trader Services enable name or service-based object discovery; however no assistance provided in correlating different method parameter representations.
Knowledge of Remote System Methods Required?	Yes. Server object must be advertised and exposed through IDL interface; server's object reference must be known to client; may use naming service or trader service to locate server application.
Modification to Existing System Required?	Yes. Must support CORBA IDL; requires client stub and server skeleton written in IDL.
Translation Methodology?	Two-step (Hardware and operating system heterogeneity resolution only). ORB translates method parameters to over-the-wire format for transmission between client and server applications; however, over-the-wire format not designed for resolving semantic heterogeneity
Capability for Application of Computer-Aid for Translation Development?	Partial. Minimal assistance provided for resolving low-level hardware and operating system heterogeneities using <i>marshal</i> and <i>unmarshal</i> process.
Support for Federation Extensibility	Partial support. Lack of practical means for extending attribute or operation parameter types under OMG IDL limits support for federation modification.
Information Exchange vs Joint Task Execution?	Both information exchange and joint task execution.

(1) Types of Heterogeneity Addressed.

Heterogeneity of Hardware and Operating Systems. One of the responsibilities of CORBA's Object Request Broker (ORB) is to provide a server method the input parameters required for its computations and to return the result of this operation to the calling client. Parameters are defined using any of OMG IDL's primitive or constructed types. Primitive types include a number of expected character, number, and logic types. Constructed types enable the creation of user-defined types by combining other types. Constructed types include an *enumerated* type, *structure* type,

union type, and *interface* type. CORBA uses a *marshaling-unmarshaling* process to transmit required parameters and return values between a client and server application. The marshaling process converts the parameter from the format used on a client to a platform-independent *over-the-wire* format for transmission between components. The unmarshaling process converts this over-the-wire format to one that is expected by the server. A similar process is utilized when returning a result to the client application. Differences in hardware platform and operating system between a client and server application are resolved using this marshaling-unmarshaling process by converting between system-specific and platform-independent formats.

Heterogeneity of Organizational Models. Although CORBA defines an object-oriented architecture, use of OMG IDL for specifying client and server interfaces enables CORBA to facilitate interoperability between both object-oriented and non-object-oriented architectures. IDL client *stubs* can be used to invoke a server's methods from either object-oriented or non-object-oriented applications. Similarly, IDL server *skeletons* enable server methods to be implemented using either object-oriented or procedural languages.

Heterogeneity of Structure, Scope, Level of Abstraction, Meaning and Temporal Validity. Communication between objects in CORBA is accomplished by one object, a client, invoking a method on another object, the server, which then performs the requested operation on a set of parameter values provided by the client and (possibly) returns a result of the operation to the calling client. The information is provided as parameters for the remote method invocation must conform to that which is expected by the server method. Both client and server must agree on the number and types of parameters used to pass information from a client to a server. The client must resolve any differences in structure, scope, level of abstraction, meaning, and temporal validity between its application's objects and the corresponding server object prior to invoking the server method. CORBA relies on the system designer to resolve these types of heterogeneity in client and server models.

Heterogeneity of Presentation. Under CORBA the client must resolve any differences in presentation between its application's objects and the

corresponding server object prior to invoking the server method. OMG's Interface Definition Language (IDL) does provide some help in resolving differences in presentation. By defining a set of *primitive* and *constructed* types and a mapping from each of these IDL types to the programming language specific types used in applications implementing a CORBA interface, IDL provides the means for eliminating problems resulting from the use of disparate data types. However, resolution of higher level differences such as domain mismatch problems, different units of measure, differences in precision, and different field lengths or variations in integrity constraints are not provided by CORBA and must therefore be addressed by the system designer using other means.

(2) Capability for Application of Computer Aid for Model Correlation. CORBA requires that method parameters be provided in the representation expected by the server object. It is the responsibility of the system designer to provide parameters that agree in scope, level of abstraction, meaning, presentation, and temporal validity with the information expected by the server. CORBA does not provide any assistance in locating client entities that might correspond to a required server parameter but differ in one of the above aspects. Similarly, CORBA does not provide any assistance in locating server methods that might utilize a specified client entity as a parameter.

(3) Required Knowledge of Remote Operations. Because information exchange and joint task execution are accomplished in a CORBA application by client invocation of server method(s), knowledge of the objects and methods available on the server is required. CORBA's Naming and Trader Services can provide assistance in locating a specified server application in which a desired object and its methods are defined; however the client must have prior knowledge of the existence and name used for the desired methods to be invoked.

(4) Required Modification to Existing System. A server application's methods are invoked by a call from a client system. This call is written using IDL and conforms to the skeleton created for the server method. If the client application does not use IDL to invoke a server's methods, then it must be modified to make it CORBA compliant.

Implementation of the server's methods can be done using any language or organizational model as long as the implementation satisfies the interface specified by the server's IDL skeleton. The server skeleton can be implemented as a wrapper surrounding a legacy method implementation, thereby eliminating the need for modification of existing server software. However, modification of the client system is required in order to add an IDL client stub used for server method invocation if it is not already CORBA compliant.

(5) Translation Methodology. In the CORBA communications model, communications between applications is facilitated by the use of an Object Request Broker (ORB). The ORB uses a marshaling-unmarshaling process to translate method parameters from the source or destination representation to an intermediate over-the-wire format for transmission across the network between client and server applications. Thus communications between components is platform independent. This intermediate representation is primarily utilized to resolve differences caused by hardware and operating systems, such as differences in word size or byte ordering. However, resolution of other differences such as domain mismatch problems, use of different units of measure, differences in precision, and different field lengths or variations in integrity constraints, are not provided by the over-the-wire format. The translation methodology for resolving such heterogeneities is not specified by the CORBA standard.

(6) Capability for Application of Computer Aid for Translation Development. Once an application component has obtained a reference to an object whose services it wants to use, it can invoke methods on that object. Generally, those methods take one or more parameters as input and return other parameters as output. As mentioned previously, CORBA's Object Request Broker (ORB) is responsible for receiving the input parameters from the component that is calling the method and for translating these parameters into a format that can be transmitted to the called object, via a process termed *marshaling*. Then, on the called object side, the ORB *unmarshals* these parameters from the transmitted format to a format that the called component

understands. Any heterogeneity in these parameters between the two systems must be resolved in order for the two systems to interoperate.

For the types of heterogeneity addressed by CORBA, this marshaling and unmarshaling process is handled completely by the ORB, entirely transparent to both the client and the server. Thus, resolution of hardware and operating system differences, as well as other low-level modeling differences, is handled automatically by the ORB in concert with the use of IDL to specify the interface between client and server systems. However, other modeling differences are not addressed by CORBA and are the responsibility of the system designer to resolve. No assistance is provided by CORBA in constructing these translations.

(7) Support for Federation Extensibility. CORBA effects joint task execution and information exchange among components of a system federation by providing the capability for one system, acting as a client, to invoke methods of another system, performing as the server. Identification of the methods exposed by a server and invoked by a client is provided by an interface defined using OMG IDL. Adding a new system to a federation can be accomplished by defining new interfaces for the services provided by that system. Existing systems would continue to use the existing interfaces to share tasks and exchange information among themselves. Existing systems would invoke methods from the newly added interfaces when desiring to interact with a new system. Conversely, a new system could interact with an existing system by invoking the method calls provided by the client stubs generated for the existing system interfaces.

Modification to existing interfaces without affecting those already in place is not as straightforward. Interface modification can be accomplished through interface extension using inheritance in OMG IDL, using the original interface to govern the original interaction between systems and the extended interface for the modified interaction. However, OMG IDL lacks practical versioning support for use in modifying types used in an interface definition. So although you can modify an interface through extension by including additional attributes or methods, OMG IDL provides no practical means for extending attribute or operation parameter types [SV02].

Although OMG IDL supports a versioning pragma that enables one to include a version number with a type, it does not solve the type modification problem. First, no version information for data types is passed between applications during method calls or replies, so the application is unable to determine what version of a type it is sending or receiving. Second, the CORBA specification does not define how the version number should be modified when a data type changes and fails to define rules for determining compatibility between different versions. This is exacerbated by failure of the CORBA standard interoperability protocol, the General Inter-ORB Protocol (GIOP), to include type information about the attributes and method parameters exchanged among components. This prevents applications from receiving types that they are not expecting or that they do not understand. [SV02]

As an alternative to the use of pragma to provide versioning support for type extension, one could instead define a new type that reflects the changed information while leaving the original type unchanged. However, modifying the interface to include both the old type and the new type definitions would require recompilation and redeployment of any applications that utilize the interface. Interface recompilation and redeployment could be avoided by the use of inheritance, defining an interface containing the new type definition as an extension to the interface containing the original type definition. However, because CORBA does not allow operation overloading, operations defined on the type in the new interface must be renamed to prevent naming collisions with the interface being extended. As additional type modifications are required, new interfaces with all-new operations to handle the new type must be derived. This approach quickly becomes unwieldy as the number of type modifications grows. [SV02]

Other approaches to versioning under CORBA have similar limitations. As a result, CORBA is considered to provide partial support for federation extensibility.

(8) Information Exchange versus Joint Task Execution. CORBA addresses both aspects of interoperability- information exchange and joint task execution. As discussed in Section II.C.1.b, CORBA's employment of IDL, together

with its communications and object model, enables one application to invoke methods defined for another. In addition to enabling joint execution of methods between two applications, this mechanism can also be utilized to exchange information between these applications. Information exchange between a client and server can be accomplished by the client invoking a *getItemFromSender* method on the server with the client supplying the values being transmitted to the sender as parameters to the method call.

2. COM, DCOM, COM+

Introduced by Microsoft in 1993, the Component Object Model (COM) is a software architecture that enables applications and systems to be built from binary components supplied by different software vendors. COM and its successor architectures Distributed COM (DCOM) and COM+ are competing technologies to the Object Management Group's (OMG's) Common Object Request Broker Architecture (CORBA). COM and its successors provide the fundamental object creation and management facilities required to enable components to interact.

COM's original function was to provide a general-purpose mechanism for component integration on Windows platforms. DCOM added support for distributed components when introduced on Windows NT in 1996 and Windows 95 in 1997. COM+ provided a unification of COM, DCOM and Microsoft Transaction Server when introduced with Windows 2000 in the spring of 2000.

a. Component Object Model (COM)

COM defines a "language-independent, object-oriented, extensible, binary interoperability standard that allows software components to communicate with each other [Kol00, p. 6]." Based on a client-server model, COM enables clients to invoke services provided by COM-compliant components, irrespective of the programming language the components are written in.

A key feature of COM is that it provides a standard that allows binary software components, supplied by different software vendors, to connect and communicate with each other. This contrasts with CORBA's approach of enabling interoperability at the source code level. Some of the key components of the COM standard which support interoperability among binary software components include:

- A provision for providing access to a software component via a strongly-typed grouping of functions termed an *interface*.
- A client-server based approach that makes the location of a function being requested by an application transparent to the calling client.
- An interface definition language for specifying and describing interfaces and objects.
- A binary standard for function calling between components.
- A base interface providing:
 - A way for components to dynamically discover the interfaces implemented by other components.
 - Reference counting to allow components to track their own lifetime and delete themselves when appropriate.
- A mechanism for uniquely identifying components and their interfaces.
- A means for object and interface reusability.
- A mechanism for identifying components and interfaces that facilitates system extension. [Kol00]

In COM, an *interface* is a collection of semantically related operations, called *methods*, which express a single functionality. An interface provides the binary standard through which clients and component objects communicate. These methods are defined in a piece of compiled code, called a *component object* (or just *object*) in COM, which provides some service to the rest of the system. A generalization of one or more interfaces that express related behavior is termed a *class*. Access to an object's methods is provided by means of an *interface pointer* that references the interface and is further described below. An interface is not a component object. A component object implements an interface and a component object must be instantiated in order for an interface to exist. Component objects can implement multiple interfaces. Interfaces are immutable, meaning that once defined, an interface cannot be changed. If access to additional methods offered by an object is desired, then a new interface must be defined.

One or more COM classes are packaged into a *server*. A server can create object instances of multiple classes, where each COM object runs inside of the server. Servers can be either *in-process* servers, where they are loaded into the same address space as the client, or *out-of-process* servers that run in another process on the same machine as the client (local) or in another process on a remote machine (remote). Remote servers are accessed using the distributed successor to COM (DCOM) that will be addressed in Section II.C.2.b. A *client* is any piece of code that makes use of another

object's services by calling methods of that object's interfaces. An important aspect of COM is that client applications do not need to know how server objects are packaged or whether the server is in-process or out-of-process. The client uses the same method to access the server in either case.

All COM objects and their interfaces are specified using the *Microsoft Interface Definition Language (MIDL)*. MIDL is an object-oriented extension of the Interface Definition Language (IDL) defined by the Open Software Foundation (OSF) for the Distributed Computing Environment (DCE). OSF IDL was originally developed for describing the interfaces, operations, and attributes for remote procedure calls (RPC) in traditional client-server applications. MIDL enables programming language independent specification of a component's interface.

COM's binary standard specifies the way that server functions must be called. A binary structure for the interface between a client and a server is defined that enables clients to utilize a server's services regardless of differences in the implementation environments of the client and server programs and how objects and their interfaces look in memory. The binary standard specifies that any interface must follow a standard memory layout. By calling the interface, a client program can obtain a pointer to a table that contains an entry for each function available via that interface. This table, called a *virtual function table (vtable)*, is an array of pointers to the object's implementations of the interface methods. A client accesses the vtable through an *interface pointer*. Therefore, client access to a server's method implementation is by means of double indirection- the client uses an interface pointer to access the vtable that in turn contains the pointer(s) to the server method implementation(s). Figure II-2 illustrates the binary structure of a COM object and the double indirection used to access a server's method implementations.

COM defines a base interface, *IUnknown*, from which all other interfaces are derived. *IUnknown* includes methods *QueryInterface*, *AddRef*, and *Release* to provide essential functionality required by all interfaces. *QueryInterface* enables a client to dynamically discover (at runtime) whether a component object supports a specified interface or not. An application will request a pointer to the interface that implements a

desired function via a call to the QueryInterface method of a component. QueryInterface will return the appropriate interface pointer and a success code if it supports that interface or an error value if it does not. Methods *AddRef* and *Release* are used to implement a manual reference counting mechanism in COM that an object uses to control its own lifetime. When a client accesses an object it uses AddRef to increase the reference counter for that object and Release to decrease the reference count when it is done with the object. When the reference count is reduced to zero, the object knows that its services are no longer needed and it therefore can delete itself.

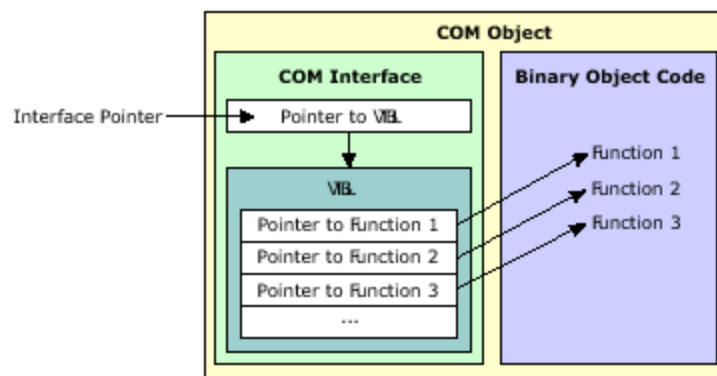


Figure II-2. Binary Structure of a COM Object (From [Kol00])

COM uses a number of globally unique identifiers in order to ensure that COM components connect to the correct component, object, or interface. Each interface is assigned a *globally unique identifier (GUID)* called the *interface ID (IID)* at development time. In addition, each COM class is assigned a *class ID (CLSID)* for the same purpose. Developers create their own GUIDs when they develop component objects and custom interfaces through the use of the *CoCreateGUID* function included as part of the COM Application Programming Interface (API). These GUIDs are embedded in the component binary and are used to dynamically ensure that no incorrect connections are made between components at bind time.

One of the primary advantages of any object model is that objects and other components can be reused and extended for use in other applications. COM enables reuse of a component object's interface through the use of *interface inheritance*. COM allows only single interface inheritance, vice multiple interface inheritance, and

does not support selective inheritance where an interface could selectively choose the methods it wants to inherit from another interface.

However, interface inheritance does not mean code reuse by inheritance, since no implementations are associated with an interface. COM does provide two other reusability mechanisms for object-level reuse. These mechanisms are *containment/delegation* and *aggregation*. In containment/delegation, one object, the *outer* object, contains another object, the *inner* object, with the outer object acting as a client to the inner object. In this way the outer object uses the inner object to implement some or possibly all of its functionality, thereby enabling the inner object to be reused by many other objects. Aggregation, on the other hand, exposes the interfaces from the inner object as if they were implemented on the outer object itself. Aggregation avoids the extra implementation overhead required by the outer object to delegate implementation of its external interfaces to a contained inner object.

The ability to modify and extend the capability of a system is generally handled via its *versioning* mechanisms. Versioning mechanisms allow you to add new features to a component, creating a new version in the process, without affecting existing clients of that component. Versioning in COM is implemented using interfaces and IUnknown's QueryInterface method.

Updating a software module is usually done to add new functionality or to improve existing functionality. In COM, since interfaces are immutable, new functionality is added to a component object by adding support for new interfaces. Since the existing interfaces don't change, components that rely on those existing interfaces are not impacted by the addition of new interfaces. Clients that know about the new functionality can use these newly created interfaces to access this functionality. The client can use IUnknown's QueryInterface method to evaluate the capabilities of a component object at runtime and when new features become available access those features through the newly created interface corresponding to those features. The procedure for improving existing functionality is even simpler. Since the syntax and semantics of an interface remain constant, the implementation of the interface can be

changed at any time, without affecting other developers' components that rely on the interface. [WK94]

COM maintains a Component Object Library to facilitate client access to server methods. When an application creates a component object, it adds the CLSID of the component object class to the Component Object Library. The CLSID is used by the Component Object Library to locate the associated server code in the registration database. COM then either launches the server code directly (if it is an executable) or loads the server code and creates an instance of the component object and returns a pointer to the requested interface back to the calling application (if it is a DLL). In either case, the calling application will use the returned interface pointer to communicate with the newly created component object. [WK94]

b. Distributed Component Object Model (DCOM)

The Distributed Component Object Model (DCOM) extends the Component Object Model (COM) to support communications among objects over a network. Whereas COM was limited to communications between processes running on the same machine, DCOM extends that capability to other machines operating across a network. DCOM uses the same methodology to communicate between networks that COM uses to facilitate inter-process communication on the same machine.

As can be seen in Figure II-3, DCOM adds a communications mechanism between client and server objects as well as a layer of middleware to connect the client and server objects to the communications mechanism. The communications mechanism is based on Microsoft's Object Remote Procedure Call (ORPC) standard. ORPC specifies how references to objects are represented, communicated, and maintained, and how calls to objects are made across the network. As shown in Figure II-3, a client and a server are connected via an underlying RPC channel. This RPC channel consists of either a local inter-process communication mechanism for client and server objects residing within the same machine, or a network protocol for client and the object residing on different machines.

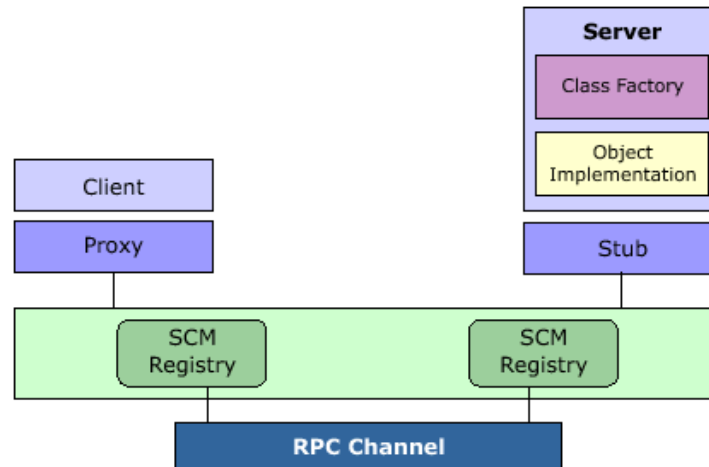


Figure II-3. DCOM Overall Architecture (From [Kol00])

Whenever a client and a server are on different processes on the same machine, or on different machines, DCOM uses the concept of *stub* and *proxy* objects to support object location transparency. The proxy object is a piece of middleware that sits between the client and the RPC channel and acts as a surrogate for the server object. It is used to package the client's method call parameters into a message buffer for transmission across the RPC channel. The method call message is received on the server side by a stub object that unpacks the received packets and forwards the client request to the appropriate object implementation on the server. If the component object being called is in-process, the call reaches the object directly using existing facilities provided by COM. Figure II-4 illustrates MIDL's use of proxy and stub constructs for achieving programming language transparency.

The Service Control Manager (SCM) is an additional piece of middleware that is used to initiate the connection between a client and a server. The SCM accomplishes this by keeping a database of class information based on registry data. Upon receipt of a client request, the local SCM looks up the desired method's object class ID (CLSID) in its registry and then takes the appropriate actions to activate the remote server. This is done by the local SCM contacting the SCM on the remote machine where the desired object resides. The remote SCM locates and launches the server and returns an RPC connection between the client and server. [DCOM96, Kol00]

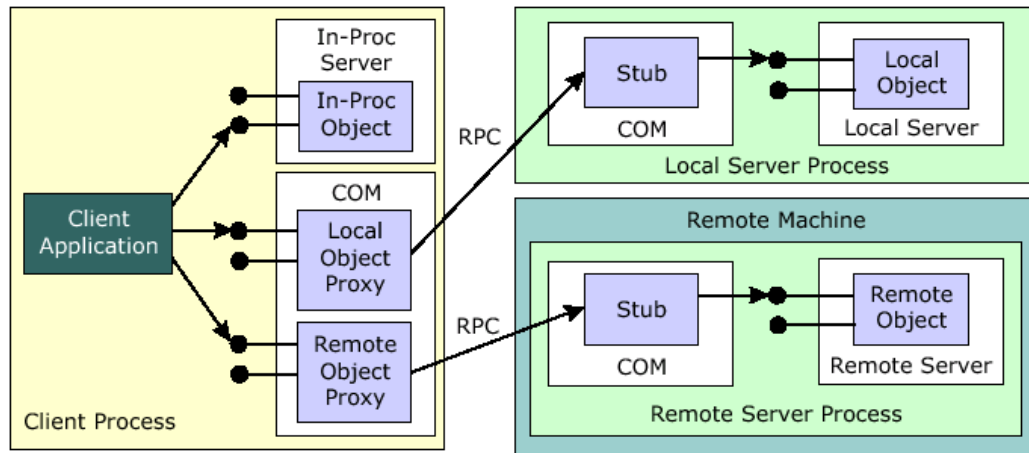


Figure II-4. Use of MIDL Proxy and Stub Constructs for Achieving Programming Language Transparency (From [Kol00])

c. *Component Object Model Plus (COM+)*

As mentioned previously, COM+ provides a unification of COM, DCOM and Microsoft Transaction Server capabilities. While adding a number of new features and services, COM+'s primary contribution regarding data interoperability is in the automation of many of the resource management tasks that developers previously had to take care of. Most notable of these tasks are:

(1) Hiding reference counting from developers. With COM and DCOM, developers were responsible for managing the lifetime of components using IUnknown's AddRef and Release methods. With COM+, reference counting is handled automatically.

(2) Largely hiding the Interface Definition Language (IDL) from developers. Rather than having to define object interfaces in terms of Microsoft's IDL (MIDL), COM+ enables developers to define these interfaces in terms of the programming language they are currently using. This greatly simplifies interface development and potentially minimizes the amount of changes required to bring legacy code into COM+ compliance.

(3) Provides a common set of types supported by all COM+ objects. Differences in the data types supported by different programming languages are problematic when attempting to create an interoperable system from a number of

heterogeneous components. COM+ provides a set of common types, shown in Figure II-5, by which components can agree for defining shared class data, method parameters, and return values. [Kol00]

Type	Size (bits)
Boolean	8
BSTR	32
Byte	8
Char	16
ClassRef (any class type)	32
COM interface pointer	32
Const (any native type)	
Currency	64
Date	64
Double	64
Float	32
HRESULT	32
Int	32
Java array (any native type)	
Long	64
Pointer (any native type)	32
Reference counted object reference	32
Safearray	
Short	16
Sized array	32
Unsigned char	8
Unsigned int	32

Figure II-5. Data Types Supported by COM+ (From [Kir97])

While not directly related to achieving data interoperability, other enhancements added by COM+, such as transaction support, load balancing, object pooling, queued components, and advanced security features improve its capability for component based system development [Kol00].

d. Evaluation of Interoperability Approach

In evaluating COM, DCOM, and COM+ against the criteria used to compare interoperability approaches, the three technologies are assessed as one as they represent the evolution of the same concept. These technologies will be referred to

collectively as COM+, except when discussing capabilities related to a specific technology in the COM+ family evolution. A summary of this assessment is contained in Table II-2 and discussed in the following paragraphs.

Table II-2. Evaluation of COM, DCOM, and COM+ Support for Resolution of Modeling Differences

Evaluation Criteria	COM, DCOM, and COM+
Types of Heterogeneity Addressed	Hardware and Operating System (partial); Organizational Models; Presentation (Partial)
Capability for Application of Computer-Aid for Model Correlation?	No.
Knowledge of Remote System Methods Required?	Yes. Method name, and type and depiction of included parameters required.
Modification to Existing System Required?	Yes. Minimal; MIDL not required by COM+, however use of client proxy still required.
Translation Methodology?	Not specified. Marshal and unmarshal routines use standard transmission format; methodology for resolution of other types of heterogeneity not specified.
Capability for Application of Computer-Aid for Translation Development?	Partial. Minimal assistance provided for resolving low-level hardware and operating system heterogeneities using <i>marshal</i> and <i>unmarshal</i> process.
Support for Federation Extensibility	Partial support. Lack of practical means for extending attribute or operation parameter types under MIDL limits support for federation modification.
Information Exchange vs. Joint Task Execution?	Both information exchange and joint task execution.

(1) Types of Heterogeneity Addressed.

Heterogeneity of Hardware and Operating Systems. The COM+ suite is considered partially successful at resolving differences in hardware and operating systems between components. Use of the Microsoft Interface Definition Language (MIDL) to specify and describe all COM objects and their interfaces, together with definition of a common set of types supported by all COM+ objects, forms the core of the COM+ family's solution for resolving hardware and operating system heterogeneity.

Coupling of the MIDL server method calls with their binary implementation through the use of virtual function tables (vtables) serves to resolve differences in the implementation environment of the client and server programs. Mechanisms for converting between system-specific type representations and the COM+ common type definition contribute further to the resolution of hardware and operating system differences. However, resolution of hardware and operating system heterogeneity is only considered partially successful due to the fact that the COM+ family is primarily designed to be supported on Windows operating systems. Microsoft has enlisted Software AG to provide COM/DCOM implementations on platforms other than Windows; however it is not certain that the full range of capabilities will be available for these other operating systems.

Heterogeneity of Organizational Models. The specification of a binary standard for defining the interface between a client and server enables the COM+ family to facilitate interoperability between both object-oriented and non-object-oriented architectures. COM's binary standard stipulates the mechanism for calling interface functions and serves to hide differences associated with heterogeneity of organizational models or programming languages used for function implementation. Functions are accessed via an interface containing a pointer to a vtable that holds a pointer to the function implementation. Thus, any programming language that can utilize a structure of pointers to explicitly or implicitly call functions can be used to write COM objects that can interoperate with other objects written to the binary standard regardless of organizational model used in the function implementation.

Heterogeneity of Structure, Scope, Level of Abstraction, Meaning, and Temporal Validity. Communication between objects in DCOM is accomplished by one object, a client, invoking a method on another object, the server, which then performs the requested operation on a set of parameter values provided by the client and (possibly) returns a result of the operation to the calling client. Communication between clients and servers on different machines is supported by DCOM using the concept of client *proxy* and server *stub* objects. The proxy and stub objects support the exchange of function calls, parameter values, and return values through a set of *marshaling* and *unmarshaling*

operations. During marshaling, the client proxy converts function parameters from the client representation to a standard format for transmission across process boundaries. Unmarshaling performs the reverse operation, converting from the standard transmission format to the server representation. Return values are marshaled by the server and unmarshaled by the client in the same manner. In order to be able to marshal and unmarshal parameters correctly, the client needs to know the exact method signature, including all data types in the parameter list, required by the server. Thus heterogeneity in structure, scope, level of abstraction, meaning, and temporal validity must be resolved by the client prior to invocation of the server method using MIDL. COM/DCOM rely on the system designer to resolve these types of heterogeneity in client and server object representations.

Heterogeneity of Presentation. The resolution of differences in presentation between systems is largely unresolved by the COM+ family. COM+ does address the problem regarding differences in data types used in different languages by defining a common set of types that are supported by all COM+ objects. However, the issue is not completely resolved since the developer may need to provide some sort of mapping between its native type definitions and those used by COM+ in order to address problems such as differences in field length and floating-point number precision [Kir97]. In addition, resolution of domain-mismatch problems, differences in units of measure, and variations in field length and integrity constraints are not addressed by COM+ and must therefore be handled by the system designer.

(2) Capability for Application of Computer Aid for Model Correlation. The COM+ family does not allow for differences in the modeling of method signatures between systems. It requires that the method name and parameters be provided in the representation expected by a server object. A client desiring to utilize a server's methods must supply the method name and parameters expected by the server. If the scope, level of abstraction, meaning, presentation, or temporal validity of information expected by the server method invocation differs from that available on a client implementation, it is the responsibility of the system designer to perform any necessary conversions between the client and server models. Since the COM+ family

does not allow modeling differences between method signatures, it does not provide any capability for correlation of client and server models of these signatures.

(3) Required Knowledge of Remote Operations. The COM+ family requires partial knowledge of the methods available for execution on a remote server. Whereas details of the method name and the type and representation of required parameters are required by a client application in order to invoke a server method, actual location of the server method is not required. Through its QueryInterface method in the IUnknown interface, COM provides a mechanism that enables clients to dynamically discover if a particular interface is supported by a component object. However, the client must have prior knowledge of the existence and name used for the desired methods in order to utilize QueryInterface to locate the specified method.

(4) Required Modification to Existing System. COM's binary standard enables objects written in different programming languages or to different organizational models to interoperate. As long as the programming language can reduce language structures to the required binary structure, then compliant objects can be used to implement a server's method calls without modification to the implementing object. However, client invocation of the desired server method is implemented using a method call written in MIDL. Therefore, modification to the client application will be necessary if its method calls were not written to be COM compliant.

COM+ eliminates the requirement for developers to implement object interfaces using MIDL, enabling them to specify the interface using whatever programming language is being used on the particular system. This minimizes the required modification to the existing system, but does not completely eliminate it, as a client proxy in the client's native language must still be provided.

(5) Translation Methodology. In DCOM, the marshaling and unmarshaling methods used by client proxy and server stub objects utilize a "flat" standard format for transmission across process boundaries. Although this standard format can handle arbitrarily complex parameter and return values, including pointers to arrays and structures or other user defined types, it primarily serves to provide the means for managing differences in word size and byte order between systems. Aside from

enabling resolution of these hardware differences between systems, this intermediate representation does not resolve other differences in presentation such as domain mismatch problems, different units of measure, differences in precision, and different field lengths or variations in integrity constraints. Use of a point-to-point or two-step translation methodology for resolving such heterogeneities is not specified for the COM+ family.

(6) Capability for Application of Computer Aid for Translation Development. As discussed earlier, client proxy and server stub objects handle marshaling and unmarshaling of method parameters. These objects automatically handle differences due to hardware heterogeneity, such as word size and byte ordering. However, other modeling differences are not addressed by COM and are the responsibility of the system designer to resolve. The COM+ family does not provide any support for application of computer aid to the development of such translation requirements.

(7) Support for Federation Extensibility. Similar to CORBA, the COM+ family utilizes a client-server model for joint execution of tasks and exchange of information among components of a federation. Identification of the methods exposed by a server and invoked by a client is likewise provided by an interface defining the system interoperation. Adding a new system to the federation is accomplished by adding new interfaces defining the services provided by that system. Existing interfaces are used to share tasks and exchange information among existing systems, while the new interfaces are utilized for interactions with the new system. Existing systems functionality is invoked by the new system using the method calls provided by the client stubs generated for the existing system interfaces.

Because COM interfaces are immutable, modification of the methods or data types used to define the information and tasks shared among systems is accomplished using interface inheritance. Although COM interfaces are immutable, objects in COM can have more than one interface. Adding new functionality to a component or modifying its existing capability is done by creating a new interface as an extension to the old one. The new interface will include the unchanged methods from the

original as well as any new or modified methods and will receive a distinct interface identifier distinguishing it from the original. Components relying on existing interfaces will continue to use them, whereas components requiring the new functionality will use the new interface.

However, just as was seen in CORBA with OMG IDL, MIDL also does not allow operation overloading. Therefore, support for type modification is similarly limited in COM+. Operations defined on a modified type in the new interface must be renamed to prevent naming collisions with the interface being extended. As additional type modifications are required, new interfaces with all new operations to handle the new type must be derived. This approach quickly becomes unwieldy as the number of type modifications grows. Therefore, the COM+ family is considered to only provide partial support for federation extensibility.

(8) Information Exchange versus Joint Task Execution. The COM+ family is primarily directed at enabling applications to share method invocation. However, as was seen with CORBA in Section II.C.1.c(7), information exchange can be accomplished by a client invoking a *getItemFromSender* method on the server, supplying the values to be transmitted to the sender as parameters to the method call.

3. Java 2 Enterprise Edition (J2EE)

a. J2EE Overview

The Java 2 Enterprise Edition (J2EE) specification “defines a Java platform with features aimed at enterprise level computing environments” [Ber00, p. 741]. It extends the Standard Edition specification primarily in the areas of security, deployment and interoperability. In support of interoperability it provides a number of distributed computing protocols and APIs that can be used in creating a system federation. Berg identifies four cornerstones for creating a distributed application:

1. Data resources
2. Naming and lookup of resources and services
3. Remote invocation or messaging
4. Transaction control [Ber00]

J2EE provides a number of APIs to address requirements for distributed computing in each of these areas:

(1) Data resources. Access to data is at the center of most multiuser computing applications. Some type of database is used to store the state of the system for just about any large system application. Standards such as SQL have provided software developers with the tools necessary to develop an application data model that is independent of the specific database used. The Java Data Base Connectivity (JDBC) API defines a standard way of accessing a relational database from a Java application, regardless of where the application and database are located. [Ber00]

The JDBC API wraps SQL and query responses in an object layer. It is based on Microsoft's API for database drivers, Open Database Connectivity (ODBC); therefore there is a great deal of conformance between JDBC and ODBC. In fact, Sun supplies a bridge that enables an application to access any data source that has an ODBC driver using the JDBC API.

A recent industry move has been toward the use of the eXtensible Markup Language (XML) for capturing data used in applications. J2EE provides three principal technologies for interacting with data stored using XML. The Document Object Model (DOM) defines an object graph structure for representing XML documents [ABK00]. Java DOM implementations include functionality for traversing and manipulating the contents of an XML document using its graph structure form. The Simple API for XML (SAX) is a set of Java packages that define an XML parser interface. SAX enables applications to process the contents of an XML document [ABK00]. The Java API for XML Processing (JAXP) uses DOM, SAX, and XSLT to support XML document processing. JAXP provides an implementation independent XML processing mechanism to parse and transform XML documents [JAXP02].

(2) Naming and lookup of resources and services. The Java Naming and Directory Interface (JNDI) provides an API for accessing name and directory services. In addition to providing a generic interface for accessing name and directory services in a uniform and product independent way, JNDI can also be used as an interface to object name services such as CORBA COS Naming.

There are two primary ways to obtain an object using JNDI: 1) an application can ask for the object by name using the lookup() method; and 2) it can

search for the object based on attributes it possesses. The lookup() method is intended primarily for name services, whereas attribute search is intended for retrieving directory services and other attribute and hierarchical types of name services. JNDI also supports referrals. A referral is a reply from a server directing a client to another server when it cannot locate an object or entry by itself but knows where it can be found. Referrals can be followed automatically without a client application having to explicitly handle a returned referral.

(3) Remote invocation or messaging. J2EE provides the capability for remote method invocation or message exchange using one or more of the following technologies. The capability for remote method invocation is provided by either Java Remote Method Invocation (RMI) or CORBA. The capability for message exchange among applications is provided by Java Message Service (JMS).

Remote Method Invocation (RMI). Java's built-in distributed object protocol, RMI, enables you to define objects that can be called remotely from other applications in a network. This capability provides the foundation for joint task execution, and consequently information exchange, among systems in a federation. RMI handles the details of packaging method parameters, sending them across a network to a remote object, unpackaging them at the destination, invoking the correct method using the passed parameters, and returning any method result back to the caller. The process of packaging parameters for transmission to a remote object is termed "marshaling" under RMI, whereas the reverse procedure of unpackaging the parameters at the destination is termed "unmarshaling."

RMI provides an analogous capability to that presented by CORBA, but does not include all of CORBA's power. RMI uses a simpler, standardized model for establishing and managing connections between a client and server.

RMI uses Java's serialization API for packaging and unpackaging data during the marshaling and unmarshaling process, respectively. Serialization is used to convert an object's state into a machine-independent encoded form that is transmitted between applications or systems. This encoded form is then reconstructed into an

equivalent object at the destination end. Any machine-specific representational differences are resolved through the marshaling-unmarshaling process.

How does RMI work? First, the system designer will define a Java interface for services that are to be made available outside a defining application. The resultant interface is then implemented with a server object class providing the required service functionality. The server object class is then compiled using an RMI compiler that generates “stub” and “skeleton” classes that provide the required linkage between remote clients and the server object.

When a client makes a remote call it is actually calling a method on the stub class that is deployed with the client code. This stub serves as a proxy for the server object, marshaling the input parameters for the server object method call onto an RMI stream and sending them to the server using an RMI communications protocol. At the server end, the skeleton code receives the RMI stream, unmarshals its contents, and invokes the method identified in the stream. Any returned value provided by the invoked method is marshaled by the server skeleton and returned to the client stub as an RMI stream. The client stub then unmarshals the returned value, creating a new Java object that is returned to the calling routing on the client side.

CORBA. Whereas Java applications can be made to interoperate with other Java applications using Java RMI, they can also interoperate with non-Java applications, and Java applications as well, using CORBA’s ORB. As stated by Berg “Java and CORBA are extremely synergistic. ... Java solves the problem of code distribution; CORBA solves the problem of intercommunication between distributed components. ... combined, they provide an architecture for creating distributed applications that can deploy themselves and run in a cooperative fashion across a network” [Ber00, p.504].

RMI’s native protocol for communications between a client and server is called Java Remote Method Protocol (JRMP). Alternatively, RMI can use CORBA IIOP to effect the marshaling and unmarshaling of method parameters between applications. The use of the IIOP communication protocol from within RMI is often referred to as “RMI-over-IIOP.” This capability enables you to utilize CORBA for

application interconnection using a programming model that mimics RMI. Using IIOP from RMI also enables you to capitalize on some of the additional features provided by IIOP such as transaction context propagation.

Java Message Service (JMS). With Java RMI or CORBA, a designer can create a system federation capable of providing joint execution of tasks and information exchange among systems. As an alternative to the remote invocation protocol provided by Java RMI, JMS provides a messaging service that can be used for information exchange among federation systems. JMS provides both a point-to-point and publish-subscribe messaging model. In the point-to-point model applications send and extract messages from named queues. In the publish-subscribe model, applications publish messages to named “topic” channels and listen asynchronously for arriving messages on these topics.

(4) Transaction control. Integration of separate applications can be done using a messaging approach or a transactional approach. In a messaging approach, the primary concern is with getting data from one system to another. A transactional approach provides for the aggregation of separate operations, possibly across different data resources, into a single transaction that preserves atomicity, data integrity, data isolation, and recoverability upon failure.

A transaction manager is “a component that coordinates the completion of transactions across multiple data resources” [Ber00, p.573]. The Java Transaction API (JTA) provides a standard Java interface that transaction managers can use to perform this coordination. When used with a transaction manager, JTA provides a transaction approach to system integration. This enables an application to create a single transaction for manipulating multiple data resources, eliminating the need for coordinating separate but related operations.

In addition to providing the JTA for managing data resource transactions, J2EE also includes the Java Transaction Service (JTS), which defines a standard Java API for interaction with CORBA’s COS Transactions service (OTS). Many of the most flexible application servers are ORB-based and use OTS for their

transaction management implementations. JTS enables the designer to include those servers in their transaction-based architecture.

b. Evaluation of Interoperability Approach

J2EE's interoperability capabilities are evaluated using the criteria specified in Section II.B. The results of the evaluation are summarized in Table II-3 and discussed in the following paragraphs.

Table II-3. Evaluation of J2EE Support for Resolution of Modeling Differences

Evaluation Criteria	Java 2 Enterprise Edition (J2EE)
Types of Heterogeneity Addressed	Hardware and Operating System (as long as there is a Java Virtual Machine implementation for the platform); Presentation (Partial. Since both client and server applications must be written in Java, common set of types available for parameter definition);
Capability for Application of Computer-Aid for Model Correlation?	No. No assistance provided for correlating different models of the real-world entities used to capture an application's problem environment.
Knowledge of Remote System Methods Required?	Yes. Client must know server's name or identifying attributes in order to acquire server object reference for remote method invocation.
Modification to Existing System Required?	Yes. Both client and server applications must be written in Java.
Translation Methodology?	Two-step (Hardware and operating system heterogeneity resolution only). Communication between objects done using Java Object Serialization; however, does not support semantic heterogeneity resolution.
Capability for Application of Computer-Aid for Translation Development?	Partial. Outside of platform independence provided by Java Virtual Machine, no assistance provided in resolving other types of heterogeneity.
Support for Federation Extensibility	Full support provided for both federation extension and modification.
Information Exchange vs. Joint Task Execution?	Both information exchange and joint task execution.

(1) Types of Heterogeneity Addressed.

Heterogeneity of Hardware and Operating Systems. Java RMI uses Java Object Serialization for passing an object from one application's address space

to another's. Serialization converts the object from the representation used by the source system to an intermediate platform independent representation in a process called *marshaling*. An *unmarshaling* process is used to reverse this procedure on the destination system, converting the object from the intermediate representation to the form used by the destination system. Thus the serialization process resolves representational differences between the source and destination system. Serialization is limited to resolving differences related to heterogeneities of hardware and operating system; resolution of other heterogeneities is left to the interoperability engineer. Serialization also requires a Java Virtual Machine (JVM) implementation for the source and destination systems.

Heterogeneity of Presentation. Since both client and server applications must be written in Java, a common set of types is available for parameter definition. Thus, presentation differences related to the use of different data types can be eliminated, assuming the designers of the interconnected systems elected to use common types in constructing the member values and methods for corresponding classes. Other causes of heterogeneity of presentation such as domain mismatch problems, the use of different units of measure, differences in precision, and different field lengths or variations in integrity constraints must be resolved by the interoperability engineer.

Heterogeneity of Organizational Models, Structure, Meaning, Scope, Level of Abstraction, and Temporal Validity. J2EE's distributed computing protocols and APIs do not provide any mechanisms for addressing the remaining types of heterogeneity- heterogeneity of organizational models, structure, meaning, scope, level of abstraction, or temporal validity. The interoperability engineer must provide his own means for resolving such heterogeneities.

(2) Capability for Application of Computer Aid for Model Correlation. The crux of the distributed computing capability provided by J2EE is the capability for an application to invoke methods of an object residing in a different address space. In providing this capability, the J2EE designers did not take into consideration potential heterogeneity in the application's model of the problem environment. It was expected that a client desiring to invoke a method from another application would

conform to the model used by the other application for defining the object's methods and parameters. Therefore, J2EE provides no assistance for correlating the possibly different models of the real-world entities used to capture an application's problem environment.

(3) Required Knowledge of Remote Operations. In order for a client to access the methods of a server, it must obtain an object reference to the Java RMI server object. This object reference can be obtained using the JNDI; however, JNDI requires either the name of the server object, or a set of attributes that can be used to determine the server object reference. JNDI provides no assistance in resolving potential heterogeneities between a client's model of the name and attribute used to identify a desired service and the name and attributes actually used by a server implementation. The client must know the actual name used by the server object or the attributes it uses to describe itself in order to obtain the server's object reference. Therefore, J2EE does require a system to have prior knowledge of a remote system's operations in order to utilize its capability.

(4) Required Modification to Existing System. If a client implementation used a different model for a server's name, attributes, or methods than that used by the server, then the client must be modified to comply with that expected by the server implementation. In addition, as the capabilities provided by J2EE can only be used with applications written in Java, if either client or server is written in another programming language then they must be modified to utilize J2EE's distributed computing capability.

(5) Translation Methodology. Java RMI uses Java Object Serialization to pass objects between systems. A serialized object is a machine-independent encoded form of the parameters and return values passed between a client and a server. The serialization mechanism implements a two-step translation methodology whereby an object's state is encoded into a machine-independent form at the source and then converted to the target system's representation at the destination. However, Java Object Serialization is limited to resolving representational differences caused by heterogeneity of hardware and operating systems only. Other types of heterogeneity are unresolved by the serialization process.

(6) Capability for Application of Computer Aid for Translation Development. Heterogeneities of hardware and operating systems are resolved under J2EE through the Java Virtual Machine and the use of Java Object Serialization for passing objects between applications. While these capabilities provide application platform independence, support for resolution of other modeling differences such as heterogeneities of organizational models, structure, presentation, meaning, scope, level of abstraction, or temporal validity is not provided under J2EE. Accordingly, facilities for applying computer aid to the development of the mechanisms required to resolve such heterogeneities is not provided under J2EE.

(7) Support for Federation Extensibility. Similar to CORBA and the COM+ family, J2EE's distributed computing capabilities can be utilized to define a client-server architecture where the methods used for effecting joint task execution and information exchange among federation components are identified by Java interfaces. Adding a new system to a federation created using J2EE's distributed computing capabilities can be accomplished by defining new interfaces for the services provided by that system. Existing systems would continue to use the existing interfaces to share tasks and exchange information among themselves. New systems could also access the capability provided by an existing system by invoking the method calls provided by the client stubs generated for that system's interfaces. Only when a new system's capabilities are required by an existing system would modification to that system be necessary in order to access the methods provided by the new system interfaces. Thus the federation can be extended without impacting original system interoperation.

In addition, J2EE provides greater extensibility during modification of the information and operations shared among systems than does CORBA or the COM+ family. Modifying a class's existing capability can be done in a manner similar to that done when adding a new system to an existing federation. In addition to providing interface extension through inheritance, classes in Java can have more than one interface. Thus changes to an existing class can be accomplished by creating a new class for the modified information as an extension to the existing one. Then, a new interface can be defined for this new class, with the new interface containing the unchanged

methods from the original class as well as any modified methods. Components relying on existing interfaces will continue to use them, whereas components requiring the new functionality will use the new interface. Also, because Java allows interface methods to be overloaded, the problem of cascading method names seen in the interface extension approach for providing type versioning in CORBA is not present in J2EE. Because J2EE provides mechanisms for both adding new systems to a federation and for modifying the existing information and operations shared among systems without impacting the interoperation of the original systems in the federation, it is considered to provide full support for federation extensibility.

(8) Information Exchange versus Joint Task Execution. As stated in Section II.C.6.f(2) above, the cornerstone of the distributed computing capability provided by J2EE is the capability for an application to invoke the methods of an object residing in a different address space. This capability for joint task execution among systems can also be used for information exchange; information to be sent from a client to a server could be included as parameter values of a *getItemFromSender* server method invoked by the client.

4. SeeBeyond Integration Suite

The SeeBeyond™ integration suite provides an architecture and set of tools designed for the integration of incompatible legacy systems, databases, packaged applications, middleware products, communication protocols, messaging standards, and data access paradigms. SeeBeyond targets the Enterprise Application Integration (EAI), Business-to-Business (B2B), and Business-to-Consumer (B2C) domains to provide an eBusiness Integration (eBI) solution to the integration of incompatible and non-interoperable business applications. The two primary components of the SeeBeyond integration suite, e*Gate™ Integrator and e*Index Global Identifier™, are discussed separately, followed by a combined assessment of the SeeBeyond suite's interoperability characteristics.

a. e*Gate™ Integrator Overview

SeeBeyond's e*Gate™ Integrator provides an open and extensible framework for eBusiness integration. e*Gate enables centralized management of the global eBusiness infrastructure, providing guaranteed delivery with packaged

transformation and application integration. e*Gate provides high-level business process management, low-level data-type conversion, and communication set-up to enable integration of incompatible legacy systems, databases, packaged applications, middleware products, communication protocols, messaging standards, and data access paradigms.

Among the capabilities provided by e*Gate is the ability to:

- Manage information exchange between legacy systems and Web Servers,
- Integrate systems based on COM, CORBA, and Java,
- Serve as a universal gateway between Oracle, SQL Server, Sybase, Informix, DB2, and older-technology databases, and
- Provide an Enterprise Integration Backbone. [EGI00]

(1) e*Gate Integrator Components. The e*Gate integrator consists of four core components, as depicted in Figure II-6: e*Ways, Intelligent Queues (IQs), Business Object Brokers (BOBs), and a central Registry.

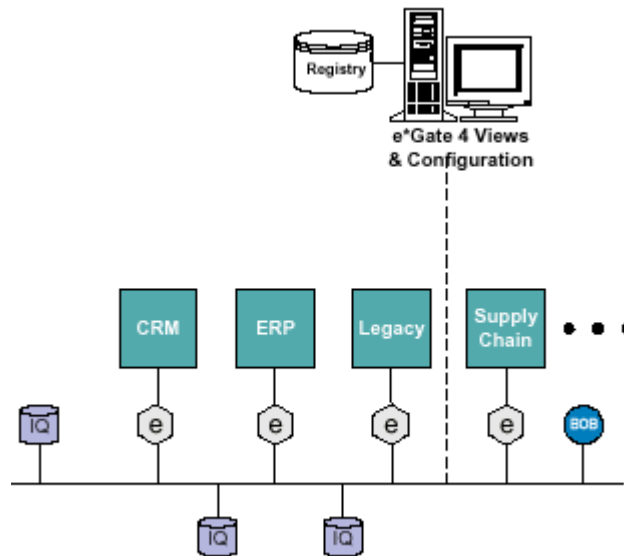


Figure II-6. e*Gate Components (From [EGI00])

An e*Way is an intelligent adapter that enables communication between a connected application, database, or similar element and other e*Gate components. It provides bi-directional, multi-threaded communication in either an event-driven or scheduled batch mode of operation. The e*Way adapter uses collaborations

defined using Java, C/C++, XSLT, or the Lisp-based Monk scripting language to provide transformations required for heterogeneous system integration. SeeBeyond makes available a number of ready-to-use database, application, file system, communication protocol, messaging system, and data format/model adapters as well as supports the capability for custom adapter generation.

Intelligent Queues (IQs) are a storage and message routing facility that provide persistent storage, guaranteed message delivery, event state support, and support for third party queues. IQs ensure that events are transmitted in the proper sequence and without duplication, even during hardware failure recovery.

A Business Object Broker (BOB) is an internal e*Way that is enabled to communicate only with IQs. BOBs can be used to implement complex, multi-step business processes. BOBs utilize routing, parallel processing, and load balancing techniques to prevent e*Way bottlenecks and to provide reliable communications across unreliable links.

The Registry is a central repository that contains the master copy of all the data processing and business rules, known as *collaborations*, as well as all of the *events* that carry information between processes. A Control Broker that updates the run-time components as information is changed replicates registry information on each platform. The Control Broker serves to insulate the platform from temporary Registry faults or communication problems. [EGI00]

(2) e*Gate Architecture. As shown in Figure II-7, e*Gate utilizes a layered architecture to separate high-level business process modeling from lower-level connectivity and translation concerns. The three layers, Views and Controls, Collaboration Logic, and Application Access are further divided into two sub-layers each as depicted.

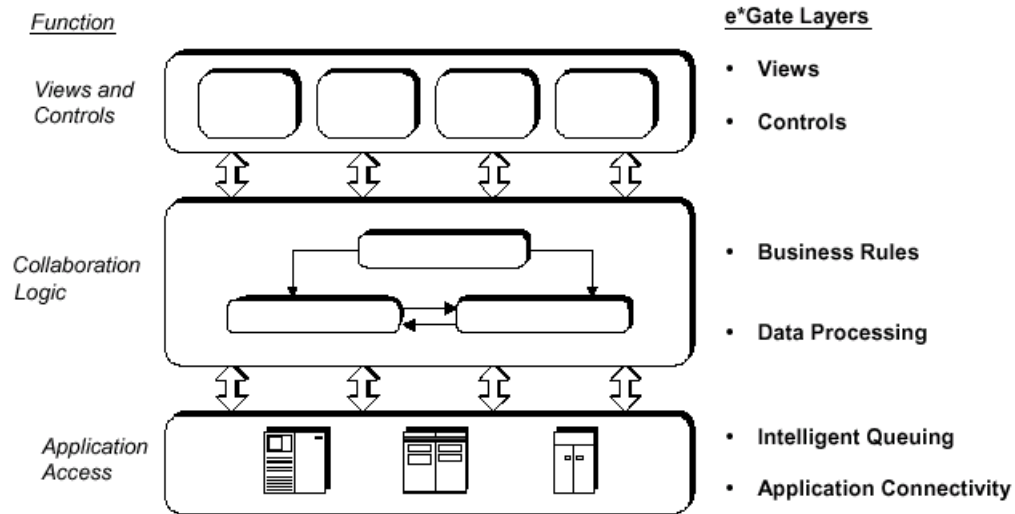


Figure II-7. e*Gate Architecture (From [EGI00])

The Views and Controls layer contains the Graphic User Interface (GUI) that enables people to use the models to run the business. This layer also includes the Registry that contains the data processing and business rules, known as *Collaborations*, and the definition of *Events*, which carry information between processes. Finally, the Views and Controls layer includes the Control Brokers residing on each host, which are used to replicate registry information on each platform.

The Collaboration Logic layer contains the Collaboration Rules Editor used to construct the data and process models that define the structure and operation of the business. It provides tools for creating a UML-based graphical model of the business processes to be automated. Analysts create Collaborations and Events that represent the business stages, transitions, and data processing activities. Collaborations use rules to identify messages, transform data and invoke APIs. The Collaboration Logic layer provides a graphical linkage between the analytical view of business rules and the technical view of messages, data and API calls necessary to implement the business rules.

The Application Access layer connects the business models to the internal and external applications. The Intelligent Queuing and Application Connectivity sub-layers route *Events* both within the e*Gate environment and to and from external applications. Intelligent Queues provide persistent recording of Event state information

necessary to ensure that Events are handled in proper sequence without risk of duplication. [EGI00]

(3) eBusiness Integration With e*Gate. eBusiness integration using e*Gate involves six steps: Model, Generate, Configure, Collaborate, Monitor, and Manage (see Figure II-8). First, a business analyst uses e*Gate to model the top-level business practices for the organization using a UML-based GUI presentation. Business rules tables associate sources with message identifiers, identifiers with field-level transformation operations, and transformed messages with destinations.

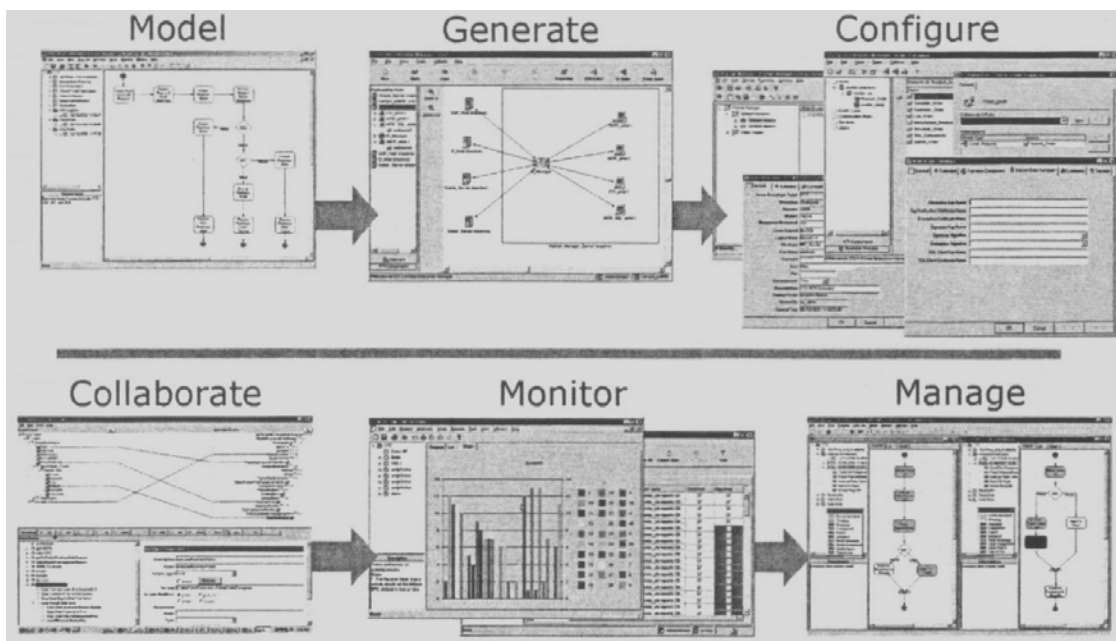


Figure II-8. e*Gate Integration Process (From [Smi01])

Then, an e*Gate companion tool, e*Insight, automatically generates e*Gate integration components. With e*Insight, users determine whether an activity should be implemented as an e*Way or a BOB in the e*Gate schema and then configure e*Gate accordingly. Users then access the Collaboration editor to create and/or modify collaborations. Reusable e*Way adapters connect e*Gate to the application to be integrated. A library of existing adapters provides compatibility with most network protocols and information environments. e*Way adapters provide translation between a connected application, database, communication protocol, etc. and the e*Gate network.

The e*Way adapter provides translation routines to convert between the associated component's format and all other required formats within the enterprise. Finally, e*Gate provides both List and Diagram views to monitor the stages that business process activities pass through as the business process instance runs, providing the analyst with the information he needs to manage the activity. [EGI00]

b. e*Index Global Identifier Overview

The e*Index Global Identifier application from SeeBeyond runs on the e*Gate Integrator platform and is designed to allow the sharing of customer information between disparate systems. The e*Index application employs a relational database of persons' records with each record automatically cross-referenced to the various local identifiers used for that person in referenced local systems. This cross-reference provides a single global identifier for a person that corresponds to the multiple local identifiers that person may have in different systems.

SeeBeyond's e*Index consists of the following key components:

- Global Person Database
- Cross-Index of Identifiers
- Real-Time Automated Matching Algorithm
- Quality Workstation for Customer Information Management

e*Index maintains a global relational database that contains a universal identifier for each person and relevant demographic information used by e*Index's automated matching process to locate the same person in multiple systems. Figure II-9 provides a view of the global identifier customer detail screen illustrating a customer universal identifier (UID) and a sample of the demographic data used for correlating customer instances.

Also depicted in Figure II-9 is the cross-index used to match a person's universal identifier in the Global Person Database to each of the local identifiers (local ID) for that person contained in systems across the enterprise. This cross-index allows systems to exchange customer information with each other without regard to differences in the local customer ID used by each system. During transit of a message through e*Gate, the local ID provided by the source system is replaced with the local ID used by the destination system, making the difference in ID's transparent to the two systems.

Detail --> INQUIRY

Identification
 UIN: 11-1111-1118
 Record Status: ACTIVE

Demographic
 Last Name: JOINSTON
 Given Name: BOB
 Middle Name/MI:
 DOB: 11/15/1962
 Gender: MALE
 Title:
 Suffix: JR
 Marital Status: SINGLE
 Religion: UN-KOWN
 Language: ENGLISH
 Race: UN-KOWN
 F/Access: NONE
 Veteran Status: UN-KOWN
 SSN: 334-53-7880
 Person Category: CUSTOMER
 Driver License: CA2374824

Address & Telephone
 Address1: 01 CLW STREET
 Address2: UNIT 212
 Home Phone: 415-345-3472
 City: SAN FRANCISCO
 State: CA
 Zip: 94114-2345
 Work Phone: 415-345-3472
 Country: UNITED STATES

Aliases

Last Name	Given Name	Middle Name/MI
JOINSTON	ROBERT	

Local ID

System	Local Identifier	Create Date
CLARIFY SERVICE	89374234329	2000/11/05
ORACLE FINANCE	09342342983728	2000/11/05
SEBEL SALES	123214387379	2000/11/05

Buttons: View List, Update, Comment, Audit Trail, Not Dup, Previous, Next, Print, Close

Ready

Figure II-9. e*Index Global Identifier Customer Detail Screen (From [EIGI00])

In constructing the cross-index, e*Index uses a real-time matching algorithm that uses fuzzy logic and statistical weighting techniques to automatically correlate local customer information with corresponding information in the global database. The matching algorithm enables a database administrator to tailor the fields used to determine a match, assign specific weights to the fields based on relative importance in deciding correspondence, and to set threshold values for automatic match determination.

The Quality Workstation application is used to facilitate managing the information within the global relational database. The workstation is used to search the database for individual records, add, delete or modify records in the database, identify and remove duplicate records, and provide an audit and reporting capability to track changes to the database. [EIGI00]

c. Evaluation of Interoperability Approach

The following evaluation of interoperability characteristics is done for SeeBeyond's eBusiness suite- their e*Gate Integrator and e*Index Global Identifier. While e*Gate provides the framework for achieving interoperability among a group of connected systems, e*Index provides additional capabilities for limited definition of a common data model for representing the real-world entities involved in system interoperation and a means for correlating this enterprise wide model with system-specific implementations. The results of the evaluation are summarized in Table II-4 and discussed below.

Table II-4. Evaluation of SeeBeyond Support for Resolution of Modeling Differences

Evaluation Criteria	SeeBeyond
Types of Heterogeneity Addressed	Hardware and Operating System - TBD; Organizational Models- partial; Structure; Scope; Level of Abstraction; Meaning; Presentation; Temporal Validity;
Capability for Application of Computer-Aid for Model Correlation?	Partial. e*Index provides correlation of customer ID's.
Knowledge of Remote System Methods Required?	Yes. (using either CORBA, DCOM, or Java RMI for remote method invocation).
Modification to Existing System Required?	No. e*Way adapter can be deployed anywhere on the network.
Translation Methodology?	Point-to-point translation definition except for e*Index' use of global identifier for Customers.
Capability for Application of Computer-Aid for Translation Development?	Yes. GUI-based Collaboration Rules editor used for translation development; reuse using library of pre-built translations.
Support for Federation Extensibility	Not determinable.
Information Exchange vs. Joint Task Execution	Information Exchange.

(1) Types of Heterogeneity Addressed. SeeBeyond's e*Gate Integrator targets the interoperation of independently developed, heterogeneous software systems. e*Gate attempts to resolve system heterogeneities that result when dissimilar operating systems, databases, communication protocols, interpretations, etc. are required to be integrated. e*Gate focuses primarily on resolving incompatibilities in databases, messaging standards, data access paradigms, and communication protocols.

Heterogeneity of Hardware and Operating Systems. While e*Gate runs on all major variations of UNIX and Windows, as well as OS/390, lack of a defined system-independent over-the-wire format such as that provided by CORBA's ORB, or a common set of types such as that defined for OMG IDL or Microsoft's IDL, makes it unclear as to what extent e*Gate supports resolution of hardware and operating system differences. From the information available in [EGI00], it appears that any differences attributable to platform or operating system must be resolved through system-specific translations provided as part of the Collaboration Rules for a specified e*Way adapter.

Heterogeneity of Organizational Models. Similarly, lack of a system- or language-independent interface definition language for specifying interactions between component systems makes e*Gate's capability for handling heterogeneities in organizational models uncertain. It does appear that e*Gate provides numerous pre-defined e*Way adapters for sharing information between Oracle, SQL Server, Sybase, Informix, DB2, IMS and older-technology databases and is thus able to resolve organizational model differences between them. Additionally, e*Gate's full support for COM/DCOM and CORBA should enable it to take advantage of those architectures' mechanisms for resolving organizational model heterogeneity.

Heterogeneity of Structure, Scope, Level of Abstraction, Meaning, Presentation, and Temporal Validity. The primary means for resolving heterogeneities among applications is through the e*Way adapter's use of pre-defined and user-defined collaborations. Collaborations contain the transformations and translations required to reconcile differences between applications. The Collaboration Rules Editor generates graphical collaboration rules based on a high level scripting language (Monk) and provides access to a library of pre-defined data and application connectivity functions.

Additionally, e*Gate provides C, C++, and JAVA Collaboration Services to enable translation specification using one of these standard programming language environments.

(2) Capability for Application of Computer Aid for Model Correlation. e*Gate, on its own, does not provide support for correlation of information being exchanged between systems. A companion product, e*Index, does provide a limited capability for correlating customers in different databases. e*Index's real-time automated matching algorithm uses fuzzy logic and statistical weighting techniques to automatically build a cross reference between a person's universal identifier and each local identifier. The e*Index matching algorithm only performs correlation of person records, although there are plans to extend the capability to other types of records. Other correlation requirements are the responsibility of the system designer. For example, in integrating databases, the system designer is responsible for determining which table in one database corresponds to which table in the other. Once this correspondence is established, e*Gate will assist the designer in identifying a mapping between table elements and in defining any translations necessary to convert between element representations.

(3) Required Knowledge of Remote Operations. e*Gate in itself does not define a client-server architecture as do CORBA, COM+, and J2EE. Therefore, e*Gate does not provide facilities for client invocation of server methods. E*Gate does provide adapters for interfacing with CORBA, COM+, and Java applications. These technologies require prior knowledge of remote system methods in order to utilize their services.

(4) Required Modification to Existing System. e*Gate uses an e*Way component to interface external applications to the enterprise. These e*Ways can be deployed on the application's platform, on another host or platform, or anywhere on the network. Therefore, modification to the existing system application is not required to utilize e*Gate's capabilities, provided that the application provides an API to expose its functionality.

(5) Translation Methodology. Although system interconnections in e*Gate are handled in a network-centric fashion, translation between different representations requires identification of the source and destination representations, effectively defining a point-to-point translation methodology and potentially requiring $n(n-1)$ translators for n different representations. e*Index's global relational database provides a universal identifier for each customer together with relevant descriptive information that serves as a common intermediate representation for customer identification; however, this capability does not extend to other entities modeled by the various systems in the enterprise.

(6) Capability for Application of Computer Aid for Translation Development. e*Gate provides the capability to translate between different data representations by enabling the designer to 1) map elements of one representation to another, and 2) define the translations required to convert one representation to the other (or choose from a pre-defined library of translations). E*Gate utilizes a GUI-based Collaboration Rules editor to assist the designer in the mapping process, and automation and reuse techniques to help with translator definition. A library of pre-built functionality is available for all levels of the integration solution (including low-level translations). In addition, the Lisp-based Monk scripting language is available for high-level definition of translations when required conversions are not available in the library.

(7) Support for Federation Extensibility. Sufficient information was not available from SeeBeyond to evaluate the support for federation extensibility provided by their products. While it is presumed that e*Gate provides some level of support for adding to or modifying existing system federations created using the product; however, details regarding any provided versioning support or other extension or modification mechanisms were not available.

(8) Information Exchange versus Joint Task Execution. e*Gate focuses primarily on resolving issues relating to data interchange. Support for joint task execution is principally handled by using adapters for interfacing COM, CORBA, and Java and applications.

5. The High Level Architecture for Modeling and Simulation (HLA)

The High Level Architecture (HLA) [HLA02] is a software architecture designed to enable individual computer simulations to be combined into larger simulations. As defined by Shaw and Garlan [SG96], a software architecture involves elements, interactions between those elements, and patterns for those interactions. In the HLA, the combined simulation system created from a compilation of individual computer simulations is termed a *federation*. The elements of a federation consist of a number of *federates*, a Runtime Infrastructure (RTI), and a common object model of data exchanged between federates, the Federation Object Model (FOM). Rules governing the interaction between federation elements are contained in the HLA standard, as are templates for defining the patterns to be followed by those interactions. I first present an overview of the elements comprising the HLA, and then discuss the HLA Specification and its contents, which provides the rules governing interactions between elements and patterns for those interactions. I then provide an assessment of HLA's capabilities, using the criteria for evaluating interoperability approaches defined in Section II.B.

a. HLA Elements

As mentioned in the previous paragraph, HLA elements consist of the individual federates comprising a federation, an RTI that enables federates to execute together as a federation, and an FOM that provides a common object model of the data exchanged between federates. A federate is an individual simulation system that forms a federation when combined with other simulations. A federate could represent one software platform such as a combat vehicle simulator, or an aggregate simulation, such as a combined battlefield simulation system used for tactics planning or training. A federate is typically larger than a common software component; it generally exists as a complete running program rather than component routines or objects in a library.

The RTI provides functions needed for simulation interoperability that apply generically to component-based simulation systems as opposed to those that are specific to a particular federate. It controls interactions between federates. It provides an interface between federates and the RTI for execution of RTI functionality. It contains network functions needed to accomplish distribution in a distributed federation. Finally, the RTI serves as an intermediary between federates, sheltering them from changes to the

RTI or to each other. The RTI is acquired from an HLA middleware vendor for use by the federation developer.

The FOM provides a common object model of the data exchanged between federates in a federation. The FOM consists of data created by the federation developer from knowledge of information to be shared among federates. The FOM is provided as a parameter to the RTI to manage a *federation execution*; a term used in the HLA to describe a group of federates executing together during a specified session. [KWD99]

b. HLA Specification

The HLA specification provides the standard for the rules governing interactions between elements of a federation and the patterns for those interactions. Specification of the initial HLA technical architecture was completed in 1996. The latest version of this specification, version 1.3, was released in April 1998 and adopted by the Object Management Group (OMG) that same year as the “Facility for Distributed Simulation Systems” standard. The Institute for Electrical and Electronic Engineers (IEEE) also approved standards P1516 (HLA Rules), P1516.1 (Interface Specification), and P1516.2 (Object Model Template) in September 2000 covering the HLA. The HLA standard detailed by the Specification consists of the following components:

- Object Model Template (OMT)- provides a meta-model that describes the allowed structure for the Federation Object Model (FOM).
- Interface Specification- specifies the interface between Federates and the Runtime Infrastructure (RTI); includes the interface the RTI presents to federates and the interface federates present to the RTI.
- HLA Rules- provide the principles and conventions that must be followed to achieve proper interaction of federates during a federation execution; HLA Rules provide the design principles for the Interface Specifications and OMT.

Details of these specification components are found in the following paragraphs.

(1) Object Model Template (OMT). The OMT prescribes the structure of the FOM for any HLA-compliant federation. A federation-specific FOM is created for each federation. The FOM describes the information that is shared among federates. Data exchange among federates is accomplished via the RTI; the FOM

prescribes the vocabulary for such data exchange. The RTI uses the FOM to achieve federate interaction.

The OMT defines two main components for use in constructing an FOM: *object classes* and *interaction classes*. Object classes are used to capture simulated entities that are of interest to more than one federate and that are expected to persist for some interval of simulated time. Interaction classes are used to represent simulated events between federates that occur at a point in simulated time but don't persist. Information regarding an object class is contained in its *attributes*. Similar information is contained in an interaction class's *parameters*. Communication between federates is accomplished via the RTI through exchange of attribute and interaction class instances.

Object and interaction classes are defined using a hierarchical structure in order to enable the FOM to change without effecting federates that depend on the original class definition. New classes are created by specializing existing classes. The HLA is designed so that an FOM can be extended without invalidating federate software written to expect the original FOM.

(2) Interface Specification. The interface specification defines the services the RTI offers to federates and vice versa. HLA services fall into six groups: 1) Federation Management, 2) Declaration Management, 3) Object Management, 4) Ownership Management, 5) Time Management, and 6) Data Distribution Management.

Federation Management services define a federation execution in terms of existence and membership. These services provide facilities for creating a federation execution and enabling a federate to join the execution or resign from it. Federation management services are also used to accomplish federation-wide operations such as synchronization between federates and federation status capture.

Declaration Management services provide the publish/subscribe mechanism by which federates share attributes and interactions. Declarations signal a federate's intent to produce or consume data and are used to transform data received by a federate. Each federate must provide a translation between its internal notion of

simulated entities and the FOM's notion. This process may be involved if a federate was not developed with the intention of HLA compliance.

Object Management services define the services used for the actual exchange of data. They are used by a federate to send and receive interactions and to register and update an object class's attributes. They are used by the RTI to send interactions, discover new objects, and to receive updates of object attributes.

A federate must own an object attribute in order to update its value. Ownership management services are used by the RTI to transfer object attribute ownership among federates. Different federates may own the various attributes of an object, and are thus responsible for updating the attributes that they own. Ownership management services govern the transfer of object ownership between federates.

Time Management services enable federates to advance their logical time in coordination with other federates in order to provide federation synchronization. These services are also used to control the delivery of time-stamped events in order to ensure proper event sequencing between federates.

Data Distribution Management services are used to control the producer-consumer relationships among federates. Whereas Declaration Management services provide the notification mechanism for federates to alert the RTI that it has data to publish or that it has a subscription request, Data Distribution Management services supply the mechanisms for providing the object and interaction class instance data to fulfill the publish-subscribe transaction.

(3) HLA Rules. HLA Rules provide the design goals and constraints for HLA-compliant modeling and simulation systems. HLA Rules include *Federation Rules* that apply to the federation as a whole and stipulate how federates must interact and *Federate Rules* that specify the interface and support that a federate must provide to the federation.

Included in the Federation Rules is the requirement for a federation to provide an FOM that stipulates the common vocabulary for the federation. The rules also require federates to exchange FOM data via the RTI, with federate-RTI interactions governed by the HLA Interface Specification. Additionally, the rules require that

simulation-specific object representations are kept in the federates and not in the RTI. Finally, Federation Rules specify that an object attribute can be owned by at most one federate at any time- this prevents problems associated with simultaneous attempts to update an attribute's value.

The Federate Rules require federates to have an HLA Simulation Object Model (SOM) documented in accordance with the HLA OMT to record simulation information that a federate might expose to the federation. These rules require that a federate comply with its SOM when sending and/or receiving attributes and interactions. They also obligate a federate to adhere to the policy specified in its SOM regarding transfer and/or acceptance of attribute ownership as well as the conditions specified for attribute update. Finally, Federate Rules stipulate that federates manage their local time in a way that will allow them to coordinate data exchange with other members of the federation. [KWD99]

c. Evaluation of Interoperability Approach

An evaluation of HLA's interoperability characteristics is done using the criteria specified in Section II.B. The results of the evaluation are summarized in Table II-5 and discussed below.

(1) Types of Heterogeneity Addressed. The RTI has no notion of the type of an attribute or parameter; it deals with them as uninterpreted sequences of bytes. If source and destination federates differ in their interpretation of transmitted data, i.e., they use different types to represent the same attribute or parameter, then the data may be interpreted incorrectly unless some type of conversion is performed between representations. In HLA, the burden of interpreting attributes and parameters is placed on the federates, requiring federation designers to agree on the interpretation selected or to provide translations to compensate for differences when they exist.

However, implementation of the HLA Interface Specification can be accomplished using other interoperability approaches. One such approach provides implementation of RTI interfaces with the Object Management Group (OMG) Interface Definition Language (IDL) API utilized by the Common Object Request Broker Architecture (CORBA). OMG IDL's *primitive* and *constructed* types could serve as a common, intermediate representation. Under this approach, federates would be

responsible for modeling object attributes or interaction parameters using OMG IDL. Differences between the federate model and the OMG IDL model relating to heterogeneity of hardware and operating systems would be resolved by the IDL implementation used by the federate.

Table II-5. Evaluation of HLA Support for Resolution of Modeling Differences

Evaluation Criteria	HLA
Types of Heterogeneity Addressed	Hardware and Operating System (using OMG IDL); Organizational Models (using OMG IDL); Presentation (Partial, using OMG IDL)
Capability for Application of Computer-Aid for Model Correlation?	No. Responsibility for resolving differences in attribute and parameter interpretation placed on federates; no assistance provided by Object Model Development Tool in establishing correspondence between interpretations.
Knowledge of Remote System Methods Required?	Not Applicable. Direct interaction between federates is not allowed in the HLA.
Modification to Existing System Required?	Yes. Modification to non-HLA-compliant federates required in order to exchange data.
Translation Methodology?	Not Specified. HLA's definition of a common object model for the data exchanged between federates (the FOM) would suggest use of a two-step translation methodology.
Capability for Application of Computer-Aid for Translation Development?	No. Responsibility for resolving differences in attribute and parameter interpretation placed on federates; no assistance provided by Object Model Development Tool for creating translations to compensate for differences in data interpretation.
Support for Federation Extensibility	Full support. Object and interaction class inheritance hierarchy enables federation to be extended or modified without invalidating existing federation software.
Information Exchange vs. Joint Task Execution?	Information Exchange.

Similarly, using OMG IDL's definition of a set of *primitive* and *constructed* types serves to eliminate problems resulting from the use of disparate data

types. However, resolution of higher-level differences in presentation such as relates to domain mismatch problems, different units of measure, differences in precision, and different field lengths or variations in integrity constraints are not resolved by IDL and must therefore be addressed by the system designer.

OMG IDL also supports resolution of differences caused by heterogeneity of organizational models. As was seen with CORBA in Section II.C.1.c(1), OMG IDL enables federates implemented in either object-oriented or procedural languages to interface with the RTI through the use of IDL's client stubs and server skeletons.

As seen in Section II.C.1.c(1), other modeling differences caused by heterogeneities of structure, scope, level of abstraction, meaning, and temporal validity are not resolved by the use of OMG IDL. These differences are the responsibility of the federates to resolve.

(2) Capability for Application of Computer Aid for Model Correlation. From experience gained during early efforts to use the architecture for achieving interoperability among simulation systems, it became evident to HLA's developers that automation was needed to support the federation development process. Three tools were identified to provide automation support to this process: 1) Object Model Development Tool, 2) Object Model Library, and 3) Object Model Data Dictionary.

The Object Model Development Tool provides automated support for developing HLA object models, for generating RTI federation execution data, and for storing and retrieving object models in the Object Model Library. The Object Model Library provides storage of Federation Object Models (FOMs) and Simulation Object Models (SOMs) to support object model reuse. The Object Model Data Dictionary supports object model standardization through maintenance of a repository of common data components for object model development.

As mentioned previously in Section II.C.5.c(1), the RTI does not consider attribute or parameter type when facilitating federate data exchange, placing the burden of attribute and parameter interpretation on the federates. Therefore, each

federate is free to use whatever model it deems appropriate for attribute and parameter representation. Potential differences between attribute and parameter models could occur, particularly when combining independently developed federates. Establishing correspondence between different models of the same data is required to enable federates to interoperate. The Object Model Development Tool could provide assistance for correlating different models of the same data on different systems. However, HLA does not provide any capability for application of computer aid for model correlation.

(3) Required Knowledge of Remote Operations. HLA provides facilities for information exchange among federation components. Such information exchange is done through interaction between a federate and the RTI; federates do not interact explicitly as they might using CORBA, the COM+ family, or J2EE. Interactions with the RTI are explicitly defined in the HLA interface specification. Direct interaction between federates is not allowed in the HLA, therefore knowledge of what operations a federate might make available for external invocation is not applicable.

(4) Required Modification to Existing System. In order to participate in an HLA federation, federates must be HLA-compliant. Federates must comply with the HLA Interface Specification in order to communicate with the RTI. They must also follow the principles and conventions for federation operation specified in the HLA rules. Modification is required for non-HLA compliant federates to exchange information using the HLA.

(5) Translation Methodology. The translation methodology used for resolving differences in interpretation of object attributes and interaction parameters among federates is not specified by the HLA. However, HLA's definition of a common object model for the data exchanged between federates (the FOM) would suggest that a two-step translation methodology be used. Using a two-step translation process, a federate would be responsible for converting the attributes and parameters from its own model to that used by the FOM.

(6) Capability for Application of Computer Aid for Translation Development. HLA places the burden of interpreting attributes and parameters on the federates. However, it does not provide any support to the developer for creating

translations to compensate for differences in interpretation of such data on different systems. While such a capability would seem appropriate for inclusion with HLA's Object Model Development Tool, it has not been provided.

(7) Support for Federation Extensibility. Under the HLA, each federation defines a Federation Object Model (FOM) that describes the data and occurrences that are shared among federates. The FOM consists of *object* classes and *interaction* classes. Object and interaction classes are organized as separate single-inheritance trees to reflect the relationships among classes. Each tree contains a single root class, *ObjectRoot* and *InteractionRoot*, which define no parameters or attributes, respectively. All other classes inherit from these root classes, with each class having exactly one immediate ancestor or superclass. The resultant inheritance hierarchies are used to protect federates from change. Federates that were written to expect and use certain object and interaction classes can continue to use them even if the FOM is modified by extending one of the existing classes. Existing federates will continue to operate using the original object and interaction classes while newer federates will use attributes and parameters from the new classes that extend the original classes.

This mechanism can be used both for adding new object and interaction classes to a federation or for modifying existing classes. The object model of the data and occurrences shared among systems (the FOM) can thus be extended or modified without invalidating federation software that was written assuming an earlier version of the FOM. Therefore, HLA is considered to provide full support for federation extensibility.

(8) Information Exchange versus Joint Task Execution. HLA is designed to support information exchange among federation components. As stated in Section II.C.5.c(3), direct interaction between federates is not allowed in the HLA, therefore precluding joint task execution among systems. In line with that prohibition, HLA objects and interactions have no behaviors associated with them in the FOM.

6. eXtensible Markup Language (XML)²

While not providing a distributed computing facility such as that offered by CORBA, COM+, or J2EE, or an architecture for combining independently developed systems such as that presented by SeeBeyond's integration suite or HLA, the eXtensible Markup Language (XML) has been publicized as a means for achieving system interoperability. XML, developed by the World Wide Web Consortium (W3C), provides a self-describing means for representing the data used by and shared among applications. It is an extension of the Standard Generalized Markup Language (SGML) and is designed to provide the flexibility and power of SGML while attempting to capture the widespread acceptance and relative simplicity of another SGML derivative, the HyperText Markup Language (HTML) [ABK00].

The cornerstone of the eXtensible Markup Language is the XML document, which consists of a sequence of data elements and element attributes with "tags" used to delimit and describe the meaning and use of the data to the user or relevant application. An XML document can consist of any permissible sequence of elements and their attributes. In order that the pertinent application might be better able to understand and utilize the data in a received document, XML utilizes a *schema* to structure and delimit the allowable values for XML documents. This schema can be in the form of a Document Type Definition (DTD) or of the newer XML Schema. Interfaced systems can utilize the DTD or XML Schema to specify the vocabulary of a system's allowable data [ABK00].

XML is actually a family of technologies. In addition to the DTD and XML Schema, XML's immediate family includes the Document Object Model (DOM), the Simple API for XML (SAX), and the eXtensible Stylesheet Language (XSL). Each of these will be discussed in the following paragraphs. First, though a quick illustration of an XML document's structure and a discussion of the key concepts of well-formedness and validity is provided.

² Portions of this material originally appeared in the thesis entitled *Integrated Development Environment (IDE) for Construction of a Federation Interoperability Object Model (FIOM)* [CY01].

a. XML basics

The best way to get a feel for XML is by viewing an example. Figure II-10 shows a simple XML document providing a citation for this dissertation.

```
<?xml version="1.0" encoding="UTF-8"?>
<reference>
  <author>Paul Young</author>
  <title>Heterogeneous Software System Interoperability Through
    Computer-Aided Resolution of Modeling Differences</title>
  <publication publicationType="Ph.D. Dissertation"/>
  <publisher>Naval Postgraduate School</publisher>
  <publisherLocation>Monterey, CA</publisherLocation>
  <publicationDate>June 2002</publicationDate>
</reference>
```

Figure II-10. Example XML Document

From the figure, XML appears to look a lot like HTML. The major difference is that XML is concerned primarily with representing the content of the data whereas HTML is more concerned with its presentation. Another difference is that XML is not constrained by a static tag set as is HTML; the XML designer is free to define tags however he sees fit. XML's tag set thus provides the added benefit of separating the model created from its view. All XML documents are properly nested (hierarchical) tree structures. This example document contains a root element `<reference>`, which contains child elements `<author>`, `<title>`, `<publication>`, `<publisher>`, `<publisherLocation>`, and `<publicationDate>`. In addition, the (empty) `<publication>` element includes an attribute *publicationType* which is used to further clarify the contents provided in the element. Notice that the XML document nicely describes the structure of the data but does not say much as to what the elements mean, other than what can be inferred by the element names.

(1) Well-formedness. A document is not an XML document unless it is well-formed, i.e., syntactically correct, according to the W3C's XML specification. This means that an ill-formed document will not be accepted for processing. This simplifies the internal code of parsers and also speeds up the processing of documents.

(2) Validity. An XML document is valid if it has an associated DTD or XML Schema and if the document complies with that schema. A schema further constrains the syntax of the XML document and also adds an implied semantics to the XML document through the terms used to define the allowable document tag sets.

b. Constraining Content

(1) Document Type Definition (DTD). A DTD specifies the logical structure of an XML document. The DTD provides a formal grammar for describing document syntax and semantics. DTD's have several characteristics that limit their effectiveness for constraining document content. First, a DTD has no capability for typing data content. DTDs treat almost all of its data as strings. Second, DTDs are not written using XML. They use their own syntax for describing the allowable content of an XML document. This disallows the use of many XML tools for displaying and manipulating information within the DTD. Third, DTDs are closed constructs; there is no simple and clear way to accomplish DTD extension, limiting the possibility of reuse between applications. Thus, DTDs are largely being supplanted with another mechanism for constraining document content, the XML Schema.

(2) XML Schema. The XML Schema addresses each of the limitations seen with the use of DTDs for constraining document content. First, XML Schema provides a number of primitive, generated, and user-defined data types for specifying data content. Primitive types reflect those found in most modern programming languages such as *string*, *boolean*, *float*, etc. Generated types build from existing types such as the primitive types or other generated types. Users can also define their own types using the primitive and generated types as building blocks in their construction. Second, an XML Schema is defined using XML syntax. This enables the capability of using existing XML parsers and other tools to construct and verify the well-formedness of XML Schemas that is not possible with DTDs. Finally, XML Schemas are extensible. A new XML Schema can be defined that extends an existing XML Schema; XML Schema can be included as part of another XML Schema, reducing the amount of rework required when defining commonly used data constructs. [BM01]

c. Programmatic Access

The Document Object Model (DOM) and Simple API for XML (SAX) were both created to serve the same purpose. Their purpose is to give one access to change and update the information stored in XML documents using any programming language. However, both of them take very different approaches in providing that access.

(1) Simple API for XML (SAX). SAX provides access to documents as a sequence of events. It works as follows. A SAX parser sequentially processes an XML document, signaling an event when a specified item such as an open tag or close tag is found. The programmer is responsible for interpreting these events by writing an XML document handler class. This handler class is responsible for specifying what action is required to be taken when a tag is encountered, such as storing an element for future reference. Document Object Model (DOM). XML only supports “has a” or “parent-child” relationships, such as a <person> may contain sub-elements of <name>, <social_security_number>, <height>, <weight>, <eye_color>, etc. This hierarchical tree structure is preserved with the Document Object Model (DOM). The DOM creates a tree of nodes based on the structure and information contained in an XML document. Interacting with this tree provides access to the information contained in the XML document. The DOM takes a generic approach, in that it will take any well-formed XML document and model it as a document object tree. Once an XML parser or other custom code has created the document object tree, access to the tree’s elements is provided for modification or deletion of existing elements, or creation of new elements, using the interfaces provided by the DOM’s API.

The choice of whether to use SAX or the DOM is dependent on how much of a document the programmer wishes to access, ease of use, and performance concerns. The SAX treats a document as a series of events, which means it can quickly and efficiently analyze large XML documents. The drawback is that the programmer has to define the data structure to hold element data. The DOM must load the entire document in memory before one has access to its data. This takes more memory and time as compared to SAX. The DOM’s strength is that the parser does almost everything, from reading the XML document in, to creating an object model of the document’s

contents, to providing a reference to this object model (a Document object) for manipulation.

d. Translations

The XML family also provides a capability for translating between different data representations via the eXtensible Stylesheet Language (XSL). XSL actually consists of three component languages: a transformation language (XSLT), an accessing language (XPath), and a formatting language (XSL-FO). As XSLF is principally concerned with the presentation of data rather than its representation or meaning, it will not be covered further.

XSLT is a high-level, declarative, XML-based language. It allows a programmer to create XSLT stylesheets that can be used to transform an XML document into *any* text-based document. Document transformation using XSL occurs as follows. First, an XSL engine is used to convert an XML document into its equivalent tree structure, which contains a number of nodes representing the elements and attributes of the XML document. Next, a stylesheet is applied to the XML document tree structure to transform the XML document to another form. XPath is used to traverse the document tree, using pattern matching to locate the document component to be transformed. Then, applying rules (templates) contained in the XSLT stylesheet, the body of the template element replaces the matched node in the source document, transforming it to its destination form.

Transformations involving other than component renaming and reordering using XSLT is limited. XSLT does have a very limited, non-standardized capability for performing functional transformations by escaping into another language such as JavaScript or Java. The W3C's XSLT Recommendation does not address any aspect of this mechanism nor does it require that an XSLT processor provide any means for performing functional transformations. This may be remedied in future versions of XSLT.

e. XML Data-Binding

Whereas XML provides extensive facilities for data definition, programming languages often provide greater capability and more efficient methods of data manipulation. XML data binding technology can be used to take advantage of an

object-oriented language's data manipulation capabilities by converting XML schema and documents into equivalent language-specific class and object definitions, respectively. Then, manipulation of the data can be performed using the programming language's native facilities, with storage and transmission of the data between applications accomplished using XML.

(1) An Object-Oriented View. If you view an XML schema in object-oriented terms, it can be equated to a class. Furthermore, an XML document, which is described and constrained by a particular schema, can be equated to an object. XML data binding is Java methodology, along with an API, that allows programs to be written that access and manipulate the content of XML documents in an object-oriented fashion.

(2) Definition. The Java Architecture for XML Binding (JAXB) Working Draft Specification [Rei01] defines XML data-binding as a facility containing two components: A *schema compiler* and a *marshaling framework*. The *schema compiler* binds components of an input schema to derived lightweight classes. A lightweight class is conceptually the same as a *Java Bean* providing access to the content of the corresponding schema component via a set of accessor and mutator (*i.e.*, **get** and **set**) methods. The derived lightweight classes will maintain all the constraints described in its corresponding schema. This ensures that when the class instance (*i.e.*, object) is unmarshaled it will not only be well-formed, but valid as well. The *marshaling framework* is a runtime API that, in conjunction with the derived lightweight classes, supports three primary operations:

- The unmarshaling of an XML document into a Java object that is an instance of a schema-derived class. This schema-derived class is composed of interrelated instances of both existing and schema-derived classes.
- The marshaling of an object back into an XML document.
- The validation of member variables against the constraints expressed in the schema.

In summary, the generated lightweight class will contain the following:

- Member variables representing the content of the input XML Schema.
- Get and set methods to access the generated member variables while maintaining constraints of the original schema.

- The unmarshal, marshal, and validate methods to convert an XML document into an instance object of the generated lightweight class and back.

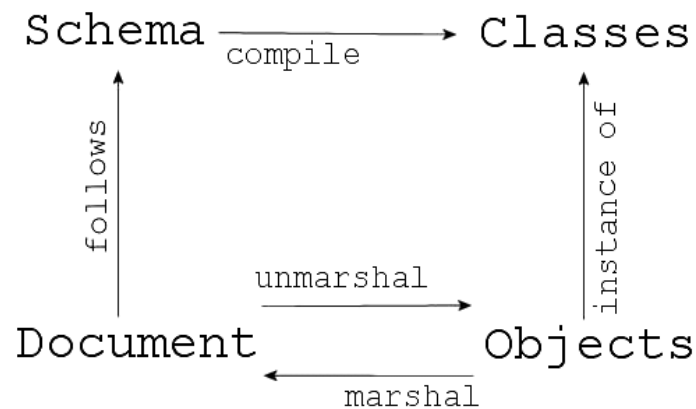


Figure II-11. XML and Java Relationships (From [Rei01]).

Figure II-11 above shows how XML Schemas, XML documents, Java classes, and Java objects are related under this framework. As can be seen, these relationships preserve equivalence, i.e., round tripping when converting between a document and an object or correspondingly when converting between a schema and a class. In other words, the unmarshaling of an XML Document and then immediate marshaling of the produced Java object(s) should result in an equivalent copy of the original XML Document.

(3) Why Use Data-Binding? The parse trees of the W3C DOM API and parser events of the SAX API are primitive, constricting, and more focused on the structure vice the content of a XML document. Also, the DOM and SAX APIs treat all data as strings requiring the casting of data to a suitable type. In contrast, data binding facilitates the direct mapping (transforming) of an XML document to objects while maintaining the constraints imposed by its corresponding schema. Thus, all the benefits, power, and familiarity of the object-oriented paradigm are available. In effect, the programmer does not have to “reinvent the wheel” in gaining access to and updating the element content within a document. The programmer has all this along with the confidence that any resulting change in state will not violate well-formedness and validity of the resulting XML document.

f. Evaluation of Interoperability Approach

XML's interoperability capabilities are evaluated using the criteria specified in Section II.B. The results of the evaluation are summarized in Table II-6 and discussed in the following paragraphs.

Table II-6. Evaluation of XML Support for Resolution of Modeling Differences

Evaluation Criteria	XML
Types of Heterogeneity Addressed	Hardware and Operating System (using XML Schema primitive and user defined types); Organizational Models (XML useable as medium for information exchange with either object-oriented or procedural languages); Structure (structural differences resolvable using XSLT); Presentation (Partial. XML Schema types enable use of common data type between representations); Meaning (differences in meaning resolvable using XSLT)
Capability for Application of Computer-Aid for Model Correlation?	No.
Knowledge of Remote System Methods Required?	Not applicable.
Modification to Existing System Required?	No. XSLT could be used in wrapper or middleware application to convert from native model of external interface to XML model.
Translation Methodology?	Not specified. XSLT could be used to define either a point-to-point or two-step translation process.
Capability for Application of Computer-Aid for Translation Development?	Partial. XSLT provides declarative approach for converting between representations. Limited capability for translations requiring more than renaming/reordering of data elements.
Support for Federation Extensibility	Partial support. Data type extension nullifies parser use for validation.
Information Exchange vs. Joint Task Execution?	Information Exchange.

(1) Types of Heterogeneity Addressed.

Heterogeneity of Hardware and Operating Systems. XML Schema provides a number of primitive, generated, and user-defined datatypes for specifying data

content. The word size and style of data format for these data types is specified by the XML Schema Datatypes specification [BM01]. As long as an XML implementation (parser, etc.) is provided for interconnected platforms and common XML Schema datatypes and constraints are used for specifying the data content used by communicating applications, heterogeneities related to differences in hardware and operating system will be resolved when using XML to exchange information among systems.

Heterogeneity of Organizational Models. XML provides a mechanism for data definition and organization that is compatible with both object-oriented and structured analysis and design approaches. Accordingly, XML is useable as an information exchange medium with either object-oriented or procedural languages.

Heterogeneity of Presentation. The primitive, generated, and user-defined datatypes specified in XML Schema provide a means for defining common data types for the information shared among systems. XML Schema datatypes additionally prescribe a means for specifying constraints on the data imposed by the problem domain. These constraints include numeric bounds, set and list ordering, permissible string representations, etc. By providing a means for defining common data types and constraints for the information shared among systems, XML provides the capability for resolving presentation differences related to the use of disparate data types. The interoperability engineer must use methods outside those provided by XML to resolve other presentation differences such as domain mismatch problems, the use of different units of measure, differences in precision, and different field lengths or variations in integrity constraints.

Heterogeneity of Structure and Meaning. When using XML to specify the data content shared by communicating applications, heterogeneities of structure and meaning between application data models can be resolved using XSLT. Conversions between models are effected using an XSLT stylesheet, which employs a declarative approach for converting between representations. Differences in structure and meaning are resolved by defining stylesheet elements that map attributes or elements in one model to their corresponding components in the other.

Heterogeneity of Scope, Level of Abstraction, and Temporal Validity. The use of XML Schema data types and XSLT do not provide the means for resolving heterogeneities of scope, level of abstraction or temporal validity. The interoperability engineer must provide other methods for resolving these types of heterogeneity.

(2) Capability for Application of Computer Aid for Model Correlation. While there are tools available for creating XSLT stylesheets that map one XML format to another, there are no tools available that help determine the correspondence between two XML formats used to model the same real-world entity in the problem environment. Determining this correspondence is the responsibility of the interoperability engineer.

(3) Required Knowledge of Remote Operations. XML does not provide the capability for remote method invocation, as do CORBA, COM+ and Java RMI. Therefore, the requirement for having prior knowledge of a remote system's operations in order to exploit their capability does not apply to XML.

(4) Required Modification to Existing System. As indicated in Section II.C.6.f(1), XSLT can be used to resolve differences in structure and meaning among independently developed systems. This is achieved by XSLT's ability to convert from one XML representation of the real-world entities whose state is shared among systems to another. Although the information exchanged among systems may not be in the form of an XML document conforming to a schema representation of the system's external interface, it may still be possible to use XML to resolve heterogeneities of structure and meaning without requiring system modification. As mentioned in Section II.C.6.d, XSLT can be used to convert from an XML representation to any other text-based document. Therefore if the federation components use a text-based mechanism for exporting or importing information (a common approach for many legacy systems), a wrapper or middleware application of XSLT could be used to convert from the text-based representation of the exported or imported information to an equivalent XML representation. Then, XSLT could again be used to resolve differences in structure and meaning between XML representations. Therefore, resolution of heterogeneities of

structure and meaning could be accomplished without modification to existing systems providing such a text-based method of information exchange.

(5) Translation Methodology. Use of XSLT to resolve heterogeneities of structure and meaning could be accomplished using either a point-to-point or two-step translation process. Translation of shared information could be accomplished using XSLT to convert directly between source and destination representations, or through the use of an intermediate representation, converting between source or destination and intermediate representations.

(6) Capability for Application of Computer Aid for Translation Development. XSLT provides the capability for resolving representational differences related to heterogeneities of structure and meaning using a declarative approach for converting between representations. Tools are available for creating XSLT stylesheets that map from one XML format to another using a graphical drag and drop interface for specifying the correspondence between the elements of the two formats. However, limited capability is provided for translations requiring more than renaming or reordering of data elements.

(7) Support for Federation Extensibility. Although XML doesn't have the same objective as the previously reviewed architectural and tool suite approaches to interoperability, it can be used in creating an interoperable federation of systems. Other approaches such as CORBA, COM+, and Java RMI are oriented around providing a distributed computing system whereby components can invoke each other's methods, whereas XML's focus is on the structured representation and definition of information. XML alone cannot be used to define an interoperable federation; however, the use of XML for describing data exchanged between components of a federation can be used to support system interoperability.

As its name implies, XML is designed to be extensible. This design feature can be used to support changes to the information shared among federation systems as additional systems are added to the federation, or extensions are made to the information exchanged among systems, or both, without adversely affecting interoperation of the original system federation. To extend the definition of an existing

data type for use in a later version, you simply add another element to the definition. Applications can use standard tools such as the DOM or SAX to manipulate the information contained in the XML definition of a data type, thus applications expecting the original data type can selectively ignore the additional data elements while applications that can make use of the additional information do so [SV02].

This methodology for data type extension is not without its limitations, as applications that validate instances of a data type against a DTD or XML Schema would fail if presented an instance of the extended data type when expecting the original data type. The choice of whether to use schema validation is up to the application, so data type extension could be accomplished by ignoring the validation feature of the XML parser. However, data type content and consistency would have to be enforced by the application, eliminating one of the benefits of XML use for data description. Since XML alone cannot be used to define an interoperable federation and data type extension cannot be accomplished in conjunction with the use of an XML parser's validation capability, XML is considered to only provide partial support for federation extensibility.

(8) Information Exchange versus Joint Task Execution. XML provides the means for representing the data used by and shared among applications. It does not provide the capability for one application to execute the methods from another as do CORBA, COM+, or Java RMI. Therefore, interaction among components of a federation using XML is limited to information exchange.

D. SUMMARY

This chapter discussed the existence of modeling differences among independently developed systems, citing the major causes of such differences as well as providing a classification of system heterogeneity. Then, a number of criteria were selected for conducting an evaluation of existing interoperability approaches in order to compare their success in resolving such heterogeneities.

These criteria were used to evaluate six of the leading approaches for achieving interoperability among independently developed systems. The first of these approaches, CORBA, provides the capability for addressing heterogeneities of hardware and operating systems, organizational models, and for partially resolving heterogeneity of

presentation. CORBA also provides the capability for heterogeneity resolution in both an information exchange and joint task execution scenario. However, CORBA's shortcomings include 1) failure to address the complete spectrum of heterogeneity; 2) lack of assistance in correlating different models of the same real-world entity on component systems; 3) lack of assistance in defining the translations required to resolve such modeling differences; 4) prior knowledge of a server's method name and the type and model of the method's parameters are needed by a client system in order to utilize their functionality; and 5) modification to existing systems not developed in compliance with the CORBA standard is required in order to enable system interoperation.

The second approach, the COM+ architecture family, provides a similar capability as that provided by CORBA. In addition, it enables interoperation among binary software components whereas CORBA addresses interoperability at the source code level. The COM+ family shares CORBA's failure to address the complete spectrum of heterogeneity and lack of assistance in correlating and resolving differences in real-world entity models. Finally, the COM+ family requires prior knowledge of remote system methods in order to utilize their functionality and requires modification to existing systems not developed in compliance with COM+ standards in order to enable their interoperation.

The third approach, J2EE, presents a competing approach to distributed computing to that provided by CORBA and the COM+ family. J2EE's strengths include its support for both information exchange and joint task execution among federation systems, its use of a two-step translation methodology for resolving heterogeneities of hardware and operating systems, and its full support provided for both federation extension and modification. Its shortcomings include 1) failure to address the complete spectrum of system heterogeneity; 2) lack of capability for assistance in establishing correspondence between different models of the same real-world entity; 3) requirement to know the server's name or identifying attributes in order to invoke the server's methods from a client; 4) requirement that both client and server applications be written in Java; 5) and lack of assistance in defining the translations needed to resolve system

heterogeneities outside of the platform independence provided by the Java Virtual Machine.

By contrast the fourth approach, SeeBeyond's e*Gate Integrator and e*Index Global Identifier, addresses each of the eight classes of heterogeneity, although some are only partially dealt with. I was unable to determine from the available literature and from communications with company representatives the extent to which heterogeneity of hardware and operating systems are able to be resolved. However it was determined that these products provide for only partial resolution of heterogeneity of organizational models. SeeBeyond does provide limited support in correlating different models of the same real-world entity on different systems. Currently this support is limited to the correlation of customer ID's through its e*Index component. SeeBeyond's eBusiness suite is primarily focused on enabling information exchange among heterogeneous systems, but the capability for joint task execution can be provided through the application of CORBA, or COM+, or Java RMI capabilities. However, use of these methods incurs the following previously mentioned limitations: 1) prior knowledge of remote system methods required, and 2) modification to existing systems needed if they are not originally developed in compliance with the specified standards. SeeBeyond does provide computer-aid to translation development; however, it does not support definition of an intermediate representation for translation, relying primarily on the point-to-point conversion between specified source and destination representation pairs.

The fifth approach, the High Level Architecture for modeling and simulation, while providing facilities for combining individual computer simulations into larger simulations, does little to address most of the limitations identified with previously evaluated approaches. HLA does present an object-oriented model for capturing data shared between systems in a federation, increasing the developer's visibility of system interaction. It also prescribes a publish-subscribe approach to data sharing, requiring interconnections between components be established only when one system has data of interest to another (or is interested in data from another system). The two primary limitations of the HLA are its capability for information exchange only, and its failure to address the possible heterogeneities among federate models of shared data. The

prohibition of direct interaction between federates and the lack of behavioral information in the Federation Object Model (FOM) precludes the use of the HLA for joint task execution. Leaving the burden of attribute and parameter interpretation on the federates and failing to provide facilities in the Object Model Development Tool to facilitate correlation of models or development of translations between different models limits HLA's support for data exchange.

Finally, the sixth approach, while not providing a distributed computing facility such as that offered by CORBA, COM+, or J2EE, or an architecture for combining independently developed systems such as that presented by SeeBeyond's integration suite or HLA, XML does provide support for achieving interoperability among independently developed systems. Of the six approaches presented, XML offers the greatest support for heterogeneity resolution, addressing, at least partially, five of the eight classes of heterogeneity defined in Section II.A.2. In addition, there are tools available that aid in the creation of XSLT stylesheets used for resolving heterogeneities of structure and meaning between application data models. Finally, XSLT can be applied using a wrapper or middleware-based approach, reducing the requirement for system modification during heterogeneity resolution. XML defines a mechanism for data definition and organization; it does not provide a method for remote invocation of methods between applications. Therefore, its support for heterogeneity resolution is limited to the exchange of information between applications. In addition, XML also shares some of the limitations found in the other approaches. These include failure to address the full spectrum of heterogeneities, lack of facilities for correlating different models of the same real-world entity, and only partial support for federation extensibility.

Common limitations of these methods include 1) lack of computer aid for determining correspondence among different models of real-world entities involved in system interoperation; 2) advanced knowledge of remote system methods required in order to access their functionality; 3) modification to an existing system required to utilize the methodology if the system is not developed in compliance with the methods' requirements; 4) no support for an intermediate representation definition, resulting in the use of point-to-point conversions between representations requiring $n(n-1)$ translations

for a federation of n systems; 5) limited capability for application of computer aid for translation development. The Object-Oriented Method for Interoperability (OOMI) presented in the following chapters of this dissertation addresses these limitations.

III. THEORETICAL FOUNDATION FOR COMPONENT SYSTEM OBJECT CORRELATION

Fundamental to the resolution of heterogeneity among a federation of independently developed systems is the identification of the real-world entities that reflect the information and operations to be shared among systems. While the information that a system wants to share with other systems is contained in the external interface of the component system, identifying the real-world entities involved in the interoperation between systems is not simply a matter of identifying the classes defined in these interfaces. Because of potential variations in what different component systems might view as important to model about an entity and in how that information might be represented, the interoperability engineer must first identify corresponding information on the different systems in the federation. Correlation of classes representing the same real-world entity must be accomplished in order to enable information and operation sharing.

This class correlation problem is similar to the query/candidate-component match problem faced in retrieving software components for reuse [Ste91, Ngu95, GNM96, Her97, ZW93, and ZW95], and the attribute correspondence problem encountered during heterogeneous database integration [CHR97, HM99, KM98, LC94, and LC00]. The approaches for solving the correlation problem in these various domains are similar. I review a number of these approaches as background for providing an understanding of the class correlation method chosen for constructing an interoperability model for a federation of heterogeneous systems, the Federation Interoperability Object Model (FIOM). Prior to this review, I first present an overview of the measures of performance used to evaluate candidate correlation methods.

A. CORRELATION MEASURES OF EFFECTIVENESS

Salton and McGill, as cited in [Ste91], point out six evaluation criteria for measuring the performance of information retrieval systems: precision, recall, effort, time, presentation, and coverage. Foremost among the effectiveness measures are precision and recall. In the class correlation context, precision is defined as the ratio of

the number of classes correctly correlated and the total number of classes correlated. Recall is defined as the ratio between the number of classes correctly correlated and the number of correct correlations possible. Recall indicates how effective the search is- what percentage of existing actual correspondences is found. Precision is an indicator of search accuracy- of the correspondences returned how many are correct. [LC00]

Effort refers to the amount of physical or intellectual labor required to correlate different classes that represent the same entity in the real world. The required correlation effort can be used to compare various correlation algorithms.

Time generally refers to the elapsed CPU or clock time required for evaluating classes to determine if a correlation exists. Time can also serve as an indirect measurement for the correlation effort required.

Presentation refers to the method in which correlation results are provided to the user. Is a ranked list of potential matches provided? Can the results be integrated with other tools attempting to solve the interoperability problem? The answer to these and related questions can be used to discriminate between potential correlation methods.

Coverage is a measure of the number of relevant classes that are contained in the external interface of component systems of the federation. In order for two systems to interoperate, a sharing of information and/or operations must occur between the systems. If the information or operations to be shared are not defined in the external interfaces of the two systems, interoperability cannot be achieved.

These criteria are used to evaluate existing correlation methods to determine their suitability for use in the Object-Oriented Method for Interoperability (OOMI) introduced in Chapter IV. Data is not available for all of the correlation measures of effectiveness on all of the correlation methods evaluated. Where available, measure of effectiveness data is provided for comparison of the data correlation methods discussed below.

B. DATA CORRELATION METHODS

Whether retrieving information from a database, locating re-usable components from a software repository, or identifying sharable information between systems in a federation, these applications share a common underlying problem. That problem is how to identify correspondences between information in order to solve the overarching retrieval or identification problem. Methods for establishing information correspondence

can be classified as classical, specification-based, or involving Artificial Intelligence (AI) [Ngu95]. An overview of some of the more relevant data correlation methodologies is provided as background for the method chosen for use in the OOMI.

1. Classical Approaches

Classical approaches include browsing, keyword matching, and multi-attribute search. A prime advantage of the classical approaches includes the availability of the information required for correlation, the relative simplicity of the different approaches, and the user's resultant ability to understand the use of the prescribed methodology. A disadvantage of the classical approaches is that they do not take component behavior into consideration, and subsequently don't achieve high values for precision and recall.

a. Browsing

A browser is a general-purpose tool for looking through collections, categories, or hierarchies of components. In its most familiar context browsers are used to manually trace through interconnected data sources on the World Wide Web. In our context of trying to determine the interoperability classes for a system federation, a user could utilize a browser to examine the component systems' external interfaces in order to identify correspondences between system classes. Browsers are commonly used and offer several advantages- 1) they are easy to understand and to use; 2) they enable a user to control the direction of search over an entire collection; and 3) they enable a user to determine dependencies between objects. However, browsers do present several disadvantages as well. First, a browser is a manual approach that is not adaptable for automatic data element correlation. Second, unless a user knows where to look for elements to correlate, misses will be likely. Third, the user won't know when to stop looking. Fourth, the user must manually inspect data elements to determine if they correspond. Finally, browsers are not suitable for large-scale repositories. [Ste91]

b. Keyword Matching

Keyword matching is a familiar approach used as the foundation for most commercial search engines. The user enters a keyword or series of keywords pertinent to the information he is trying to locate and the search engine returns all items containing some variation of the entered keyword(s). The chief advantages of keyword search are its easy implementation and conceptual simplicity for the user. Its primary disadvantage is

that the number and choice of keywords is crucial to success. It often takes an experienced user to attain the desired results. As the size of the software base increases, the effectiveness of keyword search diminishes. A tradeoff exists regarding the number of keywords used in a query- a large set of keywords results in a loss of recall, whereas a small set of keywords results in a loss of precision. [Her97]

c. Multi-Attribute Search

Multi-attribute search includes the faceted classification method of Prieto-Diez [Pri91] and the full-text information retrieval capability used by the Personal Librarian (PL) tool employed by MITRE's DELTA (Data Element Tool-Based Analysis) methodology [BFH+95]. The principle advantage of multi-attribute search is that the additional attributes used provide more information for retrieval than would be available in a pure keyword search. Its primary disadvantages are that information classification and storage are dependent on the author and/or library administrator, and that different people may classify the same information differently, reducing the effectiveness of the search.

(1) Faceted Classification. The typical classification scheme is enumerative- all possible classes are predefined and a component is classified according to the nearest fit to one of the predefined classes. In Prieto-Diaz' faceted classification scheme [Pri91], classes for categorizing a component are synthesized by selecting predefined keywords from a number of faceted lists.

A facet is formed by grouping terms into related subject areas. A faceted scheme may have several facets. To classify a component using the faceted scheme, one would select from each facet the term that best describes the component. Null values are allowed if no term in a facet describes the component. The resultant list of terms is the facet descriptor for the component.

For example, to classify the title "Structured Systems Programming" using a faceted approach and the following predefined facets,

Entities- {designs, programs, structures, systems}

Activities- {analysis, design, evaluation, programming}

would result in the descriptor {systems, programming} for our example title.

As depicted in Figure III-1, Prieto-Diaz' faceted approach consists of three principle elements- the faceted scheme, a thesaurus, and a conceptual distance graph. For each software component σ , a descriptor d_σ is composed consisting of ordered terms (T_{ij}) from each facet. The thesaurus provides a list of synonyms S_{ijk} for each concept term T_{ij} . The conceptual distance graph is used to measure similarities among terms. Two or more facet terms are related to general concepts, called notions, through weighted edges. Similarity between terms is computed by measuring the closest path between them. The conceptual distance is equal to the path weight between two terms.

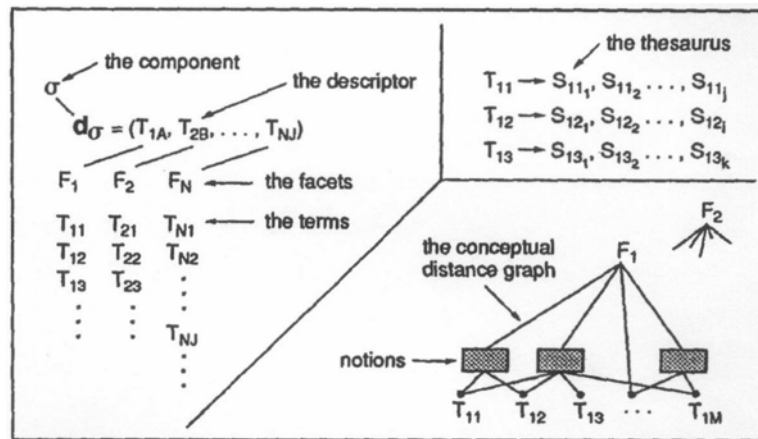


Figure III-1. Three Elements of Faceted Approach (From [Pri91])

During component retrieval, a query descriptor is created by selecting valid terms from the faceted scheme to describe the component desired. Concept ambiguity is reduced through use of a thesaurus. If there is no match for the query descriptor, a new descriptor is created using closely related terms according to the distances in the conceptual graph. A new search is then conducted using the modified descriptor. Matches on the new descriptors will retrieve components that are closely related to the component described by the original query descriptor.

Prieto-Diaz compared the recall and precision values obtained using his faceted classification scheme on a prototype system with those of a retrieval

system not organized by a classification scheme. His experiments revealed a four-fold improvement in the precision/recall ratio for the prototype using faceted classification.

Although faceted classification offers improvements in precision and recall over non-faceted retrieval systems, it does have its limitations. Chief among these limitations is the requirement for a knowledgeable librarian to classify stored components and maintain the software repository. This includes update and maintenance of the classification schemes and thesauri. Inherent in the approach is that classification is a largely manual, labor-intensive methodology. A final shortcoming is that faceted scheme effectiveness is limited for component retrieval from broad, heterogeneous collections. Its capability is geared more toward domain-specific collections where the number of facets and terms can be minimized. [Pri91]

d. Classical Approach Applicability to Interoperability Correlation Problem

The advantage of such classical approaches as browsing, keyword matching, and multi-attribute search include the availability of the information required for correlation, the relative simplicity of the different approaches, and the understandability of the prescribed methodology. The main disadvantage of the classical approaches is that they do not take component behavior into consideration, and subsequently don't achieve high values for precision and recall. Of the three approaches, keyword matching is believed to have the greatest applicability to the interoperability correlation problem. Section VI.B discusses incorporation of keyword matching into the OOMI IDE.

2. Formal Specifications

Formal specification approaches include syntax-based methodologies, semantics based techniques, and approaches using a combination of both syntactic and semantic methods. The main problems with the use of formal specifications deal with the practical problems of writing component and query specifications and the time required to conduct semantic matching. Specification writing involves formal techniques and notations that the average designer may not be familiar or comfortable with, although the techniques involved generally only require a foundation in discrete mathematics. Time problems

stem from the fact that semantic matching techniques require the use of computationally intensive theorem proving approaches.

a. *Syntax Based Approach*

The syntax based approach attempts to match a query to a candidate component based on a components' composition and structure. Zaremski and Wing introduced an approach that uses syntax information in terms of a component's signature, to locate reusable components from a software base [ZW93].

(1) Signature Matching. Zaremski and Wing [ZW93] address the correlation problem in the context of software reuse. They introduce *signature matching* as a means of locating reusable components in a library for subsequent retrieval and adaptation for use in a desired application. They consider two kinds of software components, functions and modules, for which they define the type and degree of signature matching expected for the reuse context.

A signature for a function is its type. A function type can either be a type variable or a type operator applied to other types. Type operators are either built-in operators or user-defined operators. When searching a library for a function that satisfies the needs of a specified application, the system designer defines a query in terms of the types he requires the function to contain. Matching a user-defined query to a function in the library is done based on a comparison of the types defined for the query and the types defined for a library function.

For function matching, Zaremski and Wing provide a definition for an exact match between a query and a library function as well as several relaxations that can be used to determine a partial match when an exact match doesn't hold. One type of relaxation is provided in terms of the generalization/specialization of a query where either a more general library function can be found for the query, or a function that is a specialized version of the query can be found in the library, respectively. Other relaxations allow matches to occur between a query and a library function where the order or form of a type expression defined for the function is different from that of the query. Relaxations can be combined to increase the set of library functions retrieved by a query.

Function mapping serves as the basis for providing matching between modules. The signature used for module matching is an interface, consisting of a multi-set of user-defined types and a multi-set of function types. For a match between a query and a library module to occur, there must be a correspondence between the query function types and the library module function types. In practice it is sufficient to check to see if function types match- if function types match, the modules must match- there is no need to check user-defined types. This correspondence can be either an exact match or a partial match, where not every function in the query matches every function in the library module. In addition, for each function match, the match can either be exact or relaxed as previously discussed.

The authors' work provides a set of primitive function matches that can be combined to form the most practical types of matches between a query and a set of library functions or modules. The basic technique is applicable to finding both functions and modules to match a user's query. Although their work is principally directed toward the retrieval of reusable components from a library, the same principles may be used in other applications as well, such as consolidating heterogeneous databases or resolving modeling differences in a system federation. [ZW93]

b. Semantics Based Approach

Semantics based approaches attempt to use behavior to establish correlation between elements. Behavioral information can be captured in terms of a set of conditions an element must satisfy or a set of equations describing the dynamic behavior of a component or operation. Semantics based techniques can be utilized alone or in conjunction with syntactic based methods in a multi-level approach to solving the correlation problem.

(1) Specification Matching. As discussed in [ZW93], syntactic information may be used to provide an idea of whether two components are related. However, understanding the dynamic behavior of a component, or its semantics, is key to determining whether a component provides the functionality required for use in a specified context (reuse problem), or whether two elements in a database refer to the same real-world concept (database integration problem), or whether objects may be shared between systems in a federation (interoperability problem). Zaremski and Wing

extend their syntax-based correlation efforts to address the use of semantics for determining if two software components are related. Their specification matching process supplements their signature matching process in determining the relationship between two components.

In [ZW95] the authors address specification matching as relates to both functions and modules. The dynamic behavior of a function is characterized in terms of pre- and post-condition specifications for each function asserted using first-order predicate logic. A match between two functions is determined by evaluating the logical relationship between the pre- and post-condition specifications for the two functions. There can be varying degrees of matching, from exact matches where the pre- and post-conditions of the two functions are equivalent, to implied matches where if the pre- and post-condition of one function hold (are true), then the post-condition of the other function holds also.

Function matching is defined in terms of *pre/post match* or *predicate match*. In pre/post match the pre- and post-conditions of two functions are compared; predicate match also involves the relationship between the pre-condition and post-condition for each function. For predicate match, satisfaction of the pre-condition for a function implies satisfaction of the post-condition for the function whereas in pre/post match this implication does not necessarily hold.

For pre/post matches, the strictest match is *exact pre/post match*; for this type of match the pre-conditions and post-conditions of the two functions are equivalent. In terms of measures of effectiveness, exact pre/post match results in the best values for precision (function satisfying exact pre/post match with a query exhibits the same dynamic behavior specified by the query). However exact pre/post match performs poorly in terms of recall (functions that may exhibit the desired dynamic behavior under specified conditions may be overlooked).

The next strictest match is *plug-in match*: if the pre-condition of the first function is satisfied, then the pre-condition of the second function is also satisfied, and if the post-condition of the second function is satisfied then the post-condition of first function is also satisfied. Plug-in match provides increased recall over

exact pre/post match; it will return functions that can be “plugged-in” for another function- as long as the pre-conditions of the original function holds, then a plug-in replacement will be guaranteed to meet the post-condition of the original function.

Following plug-in match in terms of relaxations is *plug-in post match*: if the post-condition of the first function is satisfied, then the post-condition of the second function would also be satisfied for functions meeting plug-in post match. Plug-in post match again increases recall; as long as the post-condition of the plug-in replacement holds, then the replacement will be guaranteed to meet the post-condition of the original function. Finally, the weakest match criterion is *weak post match*: if the pre- and post-condition of a matched function are satisfied, then the post-condition of the original function will be satisfied.

For predicate matches, the strictest match is *exact predicate match*: the predicates for two functions are logically equivalent. Exact predicate match is also less strict than exact pre/post match. The next strictest predicate match is *generalized match*; generalized match allows the specification of a matched function to be more general than the function being matched. The least strict predicate match is *specialized match*; specialized match allows the specification of a matched function to be more specific than the function being matched.

Determining a match between modules is based on the function-match foundation. There are two types of module matches- *exact module match* and *generalized module match*. For exact module match the number of functions in each module must be the same. Each function in a module must match one and only one function in the other module being compared using one of the function matches defined above. Generalized module match allows one module to match a subset of another module where, for the subset of functions matched, there is a one-to-one relationship to the functions of the other module in terms of one of the above function matches. [ZW95]

c. *Approach Using Component Syntax and Semantics*

Steigerwald, Nguyen, Herman, and Goguen et al. used a combination of syntactic and semantic search techniques for addressing the software component retrieval problem. Steigerwald [Ste91] introduced the use of algebraic specifications for defining the syntax and semantics of components in a software base in order to facilitate

automated retrieval using a similarly specified query as the search key. His *query by consistency* method was extended by Nguyen [Ngu95] and further described by Goguen et al. [GNM96], resulting in a *multi-level filtering* approach to the software component search problem. Herman [Her97] further improved Nguyen's semantic matching techniques, resulting in an increase in search precision without a loss of recall.

(1) Query by Consistency. Steigerwald [Ste91] takes a formal specification approach to retrieving source code modules from a component re-use library. His approach uses both syntactic information derived from a module's specification, and semantic information describing the dynamic behavior of a module, to retrieve components based on a user's query. Syntactic and semantic information are provided in terms of a specification written using Luqi's Prototype Specification Description Language (PSDL) [LBY88] augmented with an algebraic specification written in OBJ3. OBJ3 is an order-sorted-logic based functional programming language introduced by Goguen [GW88].

In Steigerwald's approach, reusable components are maintained in a data store together with a PSDL/OBJ3 specification defining the syntax and semantics of the component. A designer wishing to locate a component in the data store defines a query that captures the desired component structure and behavior in terms of a PSDL/OBJ3 specification. This specification serves as a key for locating corresponding components from the data store. Both the query and library component specifications are normalized, much as is done in *hashing*, in order to improve the efficiency of the search. The normalization process is performed separately for the syntactic and semantic information in order to optimize the search process. Syntactic normalization is performed using the component/query interface defined by the PSDL specification. Semantic normalization transforms the signature and axioms in the OBJ3 portion of the specification.

The syntactic and semantic normalization processes may proceed in parallel. However, syntactic matching should precede semantic matching in order to take advantage of the faster, less computationally demanding syntactic matching algorithms to reduce the candidate pool. Then, the slower, more computationally intense

semantic routines are used to further reduce the list of candidates and to rank order potential query matches. Syntactic matching provides an advantage over semantic matching in terms of speed and recall; however, semantic matching offers increased precision in locating candidate components that satisfy a query.

Search for a component can be divided into two parts: representation and search. Representation provides a model of the component sought in order to make locating the desired component easier. Search algorithms exploit the component representation to facilitate locating the component. A tradeoff exists between representation and search- the more sophisticated the representation the easier the search and vice versa.

The normalization process is used to provide the representation used for search. The ideal normalization process “would transform the axioms of two semantically equivalent objects into syntactically equivalent forms” [Ste91, p.35]. Since the axioms used to describe the semantics of a component are provided in terms of a formal language in the author’s approach, it is possible, using semantics preserving transformations, to automatically rewrite a set of axioms to an alternative form with the same meaning. The ideal approach would apply transformations to both the query and library components resulting in a *normal* form, which could then be compared for equivalency. However, due to infinite variations possible in expressing component semantics, even if you could expect to get two semantically equivalent specifications syntactically close, you would need help from a matching algorithm to determine if the two were in fact equivalent.

The search process involves the use of theorem proving techniques to show that a query specification and component specification are equivalent. The formal specification for a query and a component contain a set of axioms describing the behavior of the query or component. Taken together, the axioms of the query constitute a theory for which theorem-proving techniques can be used to show equivalency with a candidate component. However, this can be very difficult to do automatically in the general case; in addition, the process is slow and not guaranteed to terminate.

Steigerwald proposes a two-phased approach to reusable component retrieval. First, in the syntactic search phase, a comparison is made between the numbers and types of parameters found in the PSDL specification for a query and those for a candidate component. This information is used to quickly rule out components that cannot possibly satisfy the query. The author provides a set of simple tests that can be used to eliminate components in the software base as possible matches for a query. Examples of the tests include:

- If the number of input parameters in a query is not equal to the number of input parameters in a candidate component, then the component can be eliminated from consideration.
- If the number of output parameters in the query is greater than the number of output parameters in a candidate component, then the component can be eliminated from the search.
- If the query has state variables defined, but there are no state variables in the candidate component, then the component can be eliminated as a possible match.

[Ste91]

These tests provide the necessary conditions for match between a component and a query, however, they are not sufficient to determine whether a component is a syntactic match for a query.

The second phase, the semantic search phase, uses a formal specification of the syntax and semantics for both the components in the software base and the queries used to retrieve a desired component. The specification is provided using the OBJ3 algebraic specification language. Called “query by consistency” (QBC), the semantic search phase compares the specifications of a query and a component in the software base by evaluating the equivalence of reduced algebraic terms taken from the query and candidate component specifications.

Given a query specification, QBC first builds a set of example terms from the specification signature. Then, using axioms in the query, the example

terms are reduced to normal form. Term reduction involves application of the rules of a specification's axioms to a term until no further reductions are possible, resulting in normal form for the term. Similar reduction is performed on terms for components in the software base. Finally, the results of the reduction are compared in order to eliminate some candidate components and to rank order those that remain.

QBC uses Prolog as the tool to find the mappings between a query and a candidate component. Each operator definition in the signature of a query and candidate component is transformed into a set of Prolog predicate expressions. Predicate expressions derived from the component specifications are treated as Prolog facts during the mapping phase and predicate expressions from the query specification are combined to form a Prolog rule. Prolog is then used to provide a mapping from the query rule to the candidate component. With some query/component combinations, many mappings may be possible. The QBC algorithm must check every possible mapping- a task that is worse than exponential in the worst case.

Following the generation of mappings between a query and a candidate component, each mapping is evaluated to determine a score by which the potential query/candidate component matches are ranked. In order to evaluate each mapping, a test set is generated from the signature defined for the query. The test set is used to build an Input/Output (I/O) list. The I/O list consists of a list of input terms that represent the query's operators and arguments for those operators, and a list of output terms that are the result of reduction of the input terms to normal form using the axioms of the query.

The names of operators and sorts and the positions of parameters in the signature of the query will most likely be different than the corresponding operators, sorts, and parameters in the candidate component. Therefore, the terms must be transformed to the candidate component's domain using one of the mapping functions. The I/O list term output comparison will be performed in the domain of the candidate component. Therefore it is necessary to transform both the inputs and the outputs to the component domain.

Next, the reduced input and the transformed query domain output are compared using a theorem proving method referred to as *inductionless induction*. By using a sequence of rewrite rules the query input and transformed query output are compared to determine if they are behaviorally equivalent. A simple scoring mechanism is used to tally the results of the inductionless induction comparison for a particular mapping. The score given to a particular map is the ratio of the number of I/O pairs that are behaviorally equivalent to the total number of I/O pairs reduced. From these scores potential mappings between a query and a list of candidate components can be ranked.

Steigerwald does not provide any values for measuring the effectiveness of his QBC method in retrieving components from a software base. He cites the unavailability of a suitably populated library from which to determine meaningful measures. However, in the next review, Nguyen does provide a comparison of his method to Steigerwald's QBC for a small sample software base.

(2) Multi-Level Filtering. As initially reported by Nguyen [Ngu95] and reiterated by Goguen et al. [GNM96], Steigerwald's *query by consistency* method had several limitations. First, his syntactic matching process is limited to total syntactic matches (no partial matches allowed) and the use of unparameterized components. Further elimination of components not satisfying the query could be accomplished if additional information besides the number of inputs, outputs, and state variables for a component were considered. Second, the use of Prolog to find mappings between the signature of a query and a candidate component is computationally expensive- the time required to determine all possible mappings could be greater than exponential. Third, the semantic basis of QBC is not well developed. Evaluating patterns with variables gives limited information about the semantic satisfaction of a syntactic match. In his approach it is possible to have semantically equivalent specifications for which pattern evaluation would give conflicting answers, giving the appearance that a query and a component are not semantically equivalent. Finally, he provides limited support for generic components- the system does not have the ability to extract features from a user query and use them to instantiate a stored generic component in order to perform QBC. Generic parameters must be mapped to predefined sorts.

Nguyen introduced a *multi-level filtering* approach to software component search that integrates keyword, syntactic, and semantic matching techniques. Nguyen's multi-level filtering methodology provides the following improvements over Steigerwald's approach: 1) it provides a ranking of software components that potentially satisfy a query, therefore enabling partial query matches; 2) it uses a multi-level filtering approach using both syntactic and partial semantic information about components to help eliminate obvious mismatches; 3) it focuses on comparing formal specifications of components using ground equation test cases as queries, a less computationally expensive process than Steigerwald's use of Prolog; 4) it provides a method to automatically translate standard programming notation into the formal specification notation used for queries (vice having to program queries using formal specification notation); 5) it allows generic modules in the software base; 6) it addresses the structuring of the software base to increase efficiency of the search; 7) it allows user's to provide the selection criteria for controlling the search and displaying retrieved components; and 8) it provides user information to aid query reformulation for improving subsequent searches if no match to a query is found [Ngu95, GNM96].

Nguyen's software component search methodology acts as a series of increasingly stringent filters applied to a list of candidate components with regards to a user query. Components are first filtered using signature matching by comparing the signature of a query to the signature of components in the software base. This is done by mapping the type and function symbols of the query into corresponding type and function symbols of candidate components.

After signature matching, semantic matching is applied to rank components on how well they satisfy the equations in the query. In the semantic matching process, the query is used to derive equations that are logical consequences of the query specification. These equations are then translated using previously developed signature matches to a set of equations whose proof is attempted using the candidate specifications. Candidate components are then ranked according to the success of the candidate specification proof process.

The syntactic matching approach uses three levels of syntactic filtering: 1) *profile filtering* is used to partition the software base, as an aid to signature matching, 2) *keyword filtering* reduces the candidate component set, and 3) *signature matching* “finds the maps that translate the type and function symbols of the query into the corresponding type and function symbols of the candidate components” [Ngu95 p.15].

For signature matching, each component in the library has an associated algebraic specification consisting of a signature for the component and a set of equations used to specify the component’s behavior. The component signature consists of a set of sorts (types) defined for the component and a set of functions on those types. The set of equations for a component specifies properties that the functions of the component should satisfy. Signature matching attempts to find a mapping between the sorts and functions of a query and the sorts and functions of a library component.

The effort required for matching the sorts and functions of a query to those of a candidate component is computationally expensive. Profile matching is introduced as a means for reducing the library component search space in order to speed up the signature matching process.³ The number and organization of sorts in a component’s functions are used to define a profile for each operation. Nguyen defines an operation profile as:

- “1. The first integer is the total number of occurrences of sorts.
2. If the total number of sort groups, N , is greater than 0, then the second to $(1 + N)^{\text{th}}$ integers are the cardinalities of the sort groups, in descending order.
3. The $(2 + N)^{\text{th}}$ integer is the cardinality of the unrelated sort group.
4. The $(3 + N)^{\text{th}}$ integer is:
 - 0 if the value sort is different from any of the argument sorts; and
 - 1 if the value sort belongs to some sort group” [Ngu95 p.20].

³ Steigerwald’s approach looked at all possible matches between a query and a component signature; Nguyen introduces profile matching in order to reduce the number of possible mappings that must be evaluated by signature and semantic matching algorithms.

The composition of all the operation profiles for a component provides a profile for that component. The profiles for a group of components in a library are used to separate the library into partitions of components containing the same profile. By comparing profiles for a query and a component it can be determined whether the component matches the query. The degree of match between a component and query is given as a *ProfileRank*, which provides a numerical value based on the number of operational profiles for a query that match the operational profiles for a component, given as:

$$\text{ProfileRank}(B_Q, B_C) = |B_Q \cap B_C| / |B_Q|$$

where B_Q represents the profile for a query and B_C a component profile. Again, *ProfileRank* provides a measure of recall for the profile matching algorithm.

The keyword rank function is used to measure how close the keywords of a query, Kw_Q , are to the keywords of a component, Kw_M :

$$\text{KeywordRank}(Kw_Q, Kw_M) = |Kw_Q \cap Kw_M| / |Kw_Q|$$

The keyword rank function measures recall by determining the ratio of the number of keywords present in both the query and a candidate component to the number of keywords in the query.

Finally, the signature-matching algorithm is performed on only those components in the partition whose profile matches that of the query. Signature matching determines how close the signature of a query matches the signature of a component. The result of executing the signature matching algorithm for a candidate component is a *SignatureRank* for the component, defined as the ratio of the number of elements in the signature of the component that match the signature of the query to the number of elements in the signature of the query.

$$\text{SignatureRank}(V, Q) = |V \cdot \Sigma| / |Q \cdot \Sigma|,$$

where “ $Q \cdot \Sigma$ is the signature of the query and $V \cdot \Sigma$ is the subsignature of $Q \cdot \Sigma$ that is actually matched by V ” [Ngu95 p.28].

The choice of software base components that match a query can be further refined using semantic filtering. A semantic validation procedure tests equations from the query for satisfaction in the candidate component specification. First, a set of *ground equations* is formed from the query specification. *Ground equations*, that is equations whose terms have no variables, are used to reduce the complexity required for checking candidate component matches to a query. Then, the query ground equations are tested for satisfaction in the candidate component specification as follows. First, the query ground equations are translated using the signature match information between the query signature and candidate component signature determined during the signature match process. Then, the translated ground equations are evaluated against equations in the candidate component specification. If the translated ground equations are satisfied, i.e., evaluation of each equation results in the identity relation, then the candidate component satisfies the query.

Using the results of the semantic validation procedure, a ranking can be computed for candidate components in the software base that satisfy a query. The ranking is computed on an operation by operation basis, based on whether a signature match is defined for the operation and if so whether the semantic validation procedure results in a successful match between the query and the component operation. The ranking takes into consideration the possibility of partial satisfaction of a query's ground equations by a component. Finally, the *SemanticRank* of the component is computed as the sum of the ranking for the operations defined for the component. Then, given the *SemanticRank* of a component, the author's compute its overall *ComponentRank* as follows:

$$\text{ComponentRank} = \text{KeywordRank} * \text{ProfileRank} * \text{SignatureRank} * \text{SemanticRank}$$

Thus, the designer is provided an ordered list of components from which to evaluate for possible application in his implementation.

Nguyen provided an evaluation to compare the performance of Steigerwald's signature-matching method to his approach that added profile matching to the signature matching process. From his limited evaluation of matching a query against a library of six candidate components, he reported a fifty-six-fold improvement in the

time required for semantic matching with profile matching included vice that required when the algorithm did not include profile matching.

Nguyen also performed an evaluation of the precision and recall values attainable when searching a sample library for components matching a given series of queries. When searching for components that were an exact match to the given query, his approach yielded very high precision values (1.0 average) but low recall numbers (average of .58). When the search was expanded to include partial matches as well, recall values improved to an average of 1.0 while precision values fell to .36 on average. A third scenario, which he termed the “Medium KPS scenario”, provided more balanced precision and recall results, resulting in average values of .81 and 1.0 respectively. Nguyen found that his exact match search compared favorably with Prieto-Diaz’s faceted approach [Pri91] for precision but was outperformed slightly for recall. However, he found that his Medium KPS scenario had a better recall performance but a lower precision than the faceted approach [Ngu95].

(3) Improved Multi-Level Filtering. Nguyen’s multi-level filtering approach combines keyword matching, syntactic matching and semantic matching to retrieve reusable components from a software base. Syntactic matching is further decomposed into two phases: profile filtering and signature matching. Herman [Her97] introduces improvements to the resolution of syntactic profiles in order to increase precision during profile filtering, without a loss in recall. In addition he presented improvements to the internal representation of syntactic profiles in order to improve the time and space requirements for profile representation. Finally, he provided improvements to the signature matching process that reduced the time required to find valid signature maps.

Herman’s multi-level filtering model is presented in Figure III-2. The multi-level filtering process consists of two main activities: syntactic matching and semantic matching, with syntactic matching further broken down into profile filtering and signature matching. The purpose of the multi-level filtering approach is to eliminate as many candidate components in response to a query using earlier, less computationally expensive approaches before producing a ranked set of potential matches using the

semantic matching process. He also enables manual inspection of the intermediate results in order to further eliminate candidates from consideration.

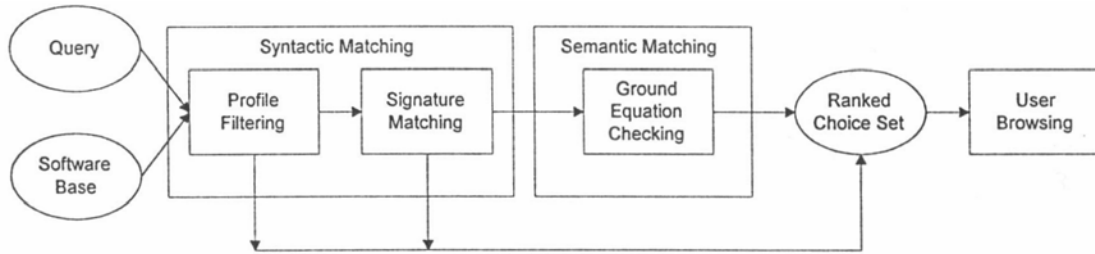


Figure III-2. Multi-level Filtering Model (From [Her97])

Herman proposed improvements to the definition of an operation profile defined by Nguyen in order to increase the precision offered by his approach. His proposal included improvements to both the resolution of the profile and the space and time required to store a profile and search through a collection of profiles to determine a match to a query.

Improvements in profile resolution provide the ability to better distinguish between syntactically similar software components. Herman offered two approaches to increasing the resolution of a component profile: 1) increase the number of possible discriminates for a profile property and 2) increase the number of properties included in a profile.

Herman's first improvement to Nguyen's profile definition was to modify the value of the last integer in the profile to represent the number of occurrences of the value sort in the operation's signature. Nguyen had defined the last integer as a true/false indicator of whether the value sort was also included as an argument sort for the operation. His second improvement was to include an additional property in the profile to capture the number of times the type sort of a component (assuming the component represented an abstract data type) was present in an operation. The third profile resolution improvement was the addition of a count of the frequency of occurrence of five predefined sorts (boolean, character, string, integer, real) in the

signature of an operation. In summary, Herman's improvements resulted in the following update to the profile definition provided in [Ngu95]:

1. The first integer is the total number of occurrences of sorts.
2. If the total number of sort groups, N , is greater than 0, then the second to $(1 + N)^{\text{th}}$ integers are the cardinalities of the sort groups, in descending order.
3. The $(2 + N)^{\text{th}}$ integer is the cardinality of the unrelated sort group.
4. The $(3 + N)^{\text{th}}$ integer is the number of occurrences of the type sort in the operation's signature (for abstract data types).
5. The $(4 + N)^{\text{th}}$ through $(8 + N)^{\text{th}}$ integers are the number of occurrences of the predefined sorts (boolean, character, string, integer, real) in the operation's signature.
6. The $(9 + N)^{\text{th}}$ integer is the number of occurrences of the value sort in the operation's signature. [Her97]

In addition to the profile resolution improvements offered by Herman, he also enhanced the profile storage and retrieval efficiency through two measures: 1) the use of a large integer to represent a profile and 2) the use of a profile lookup table for storing component profiles. The large integer representation uses the digits of a large (double precision) integer to represent an operation profile vice the set of integers used by Nguyen. The profile lookup table assigned a unique integer lookup ID to an operation profile so that a component profile could be represented as a collection of lookup ID's vice a multiset of operation profiles, each of which consists of a sequence of integers.

In addition to providing syntactic profile resolution enhancements and time and space improvements in the storage and use of syntactic profiles, Herman reduced the time required to find valid signature maps by improving the signature matching process. These enhancements include: 1) ordering profiles lexicographically to help constrain the search during operation matching; 2) reduction of the search space by attempting to match operation outputs before considering input sorts; and 3) using the

preservation rules of predefined types to reduce the number of input sort permutations that must be considered.

By ordering profiles lexicographically, his approach attempts to match the sorts of smaller operations before matching the sorts for larger operations. This potentially results in an improvement of the matching process because smaller operations constrain the number of matching possibilities and therefore can contribute to an early reduction of the search space.

By matching output sorts prior to input sorts, the numbers of input sorts that have to be matched in an operation are reduced. If an output sort for a query is matched to a specific output sort in the candidate component, then that matching must hold for the query sort when used as an input parameter as well. Operations where the matching does not hold can be eliminated from consideration. In addition, once a query's output sort is matched to the output sort of a candidate, that sort cannot be matched to a different output sort in the candidate. This further reduces the search space.

Finally, the type preservation rules of predefined types enables further pruning of the number of potential matches that must be considered by the signature matching and semantic matching routines. The type preservation rules require that in order for a query operation to match a candidate component operation, the input type of a query must be either the exact type or a subtype of the input type in the candidate, and the output type of the query must be either the exact type or a supertype of the output type in the candidate. Failure to meet this requirement will eliminate a potential match from consideration.

Experimental validation of Herman's suggested improvements supported his hypothesis in four areas: 1) software base resolution; 2) profile filtering performance; 3) profile improvements' impact on signature matching; and 4) signature matching performance enhancements. A basic premise of the recommended profile enhancements is that increasing the profile resolution will enhance the resolution of the software base by increasing the number of partitions in the software base and consequently reducing the number of components per partition. As a result, query match precision should increase without reducing recall. Herman's measurement of the number

of software base partitions shows a 65% gain in the number of partitions due to implementation of his recommended profile enhancements.

As a measure of profile filtering performance, Herman hypothesized that his recommended profile enhancements would result in a reduction in the number of components returned as candidate matches to a query. Experimentation validated this hypothesis, resulting in as much as a 79% decrease in the number of returned components when all three profile enhancements were implemented.

A concise quantification of the effect profile improvements had on signature matching was not made in the thesis. However, a look at one query match attempt showed that profile improvement implementation enabled a valid signature match to be obtained where it was not attainable without the suggested profile improvements and that generally, matches using the improved profile fared at least as good as without the improvements.

An evaluation of the signature matching performance enhancements was conducted by counting the number of nodes that passed or failed the output matching and predefined type matching tests. Failed nodes represent nodes that are pruned, so an increase in the number of failed nodes indicates improved signature matching performance. Herman provided a measurement of the number of nodes that passed and failed the enhanced signature matching tests for each of the various profile resolution improvements. However, it is not possible to discriminate between failure increases resulting from the signature matching enhancements and those resulting from the profile resolution improvements. [Her97]

d. Formal Specifications Applicability to Interoperability Correlation Problem

Zaremski and Wing [ZW93, ZW95], Steigerwald [Ste91], Nguyen [Ngu95], Goguen et al. [GNM96], and Herman [Her97] introduced the use of formal specifications in an attempt to exploit behavioral aspects for correlation. As mentioned earlier, the main barrier to the use of formal specifications involves the practical problems of writing component and query specifications and the time required for conducting semantic matching. Perhaps as a result of the difficulty involved in writing formal specifications, the legacy systems targeted for integration generally lack any type

of formal specifications for describing system behavior. However, type signatures are generally available, and aspects of the syntactic matching parts of this group of methods could be applied. On the other hand, the strengths of syntactic matching are not well exploited in typical interoperability applications because the explicit type structures in these applications are not very rich- fields are usually limited to numbers and strings and there is not enough structure to provide much discrimination between alternatives. Therefore, in order to best take advantage of this approach for determining class correspondence, formal specifications would need to be created for systems lacking their use. The time and effort required to augment existing systems with formal specifications describing their behavior would outweigh any savings gained from the introduction of computer aid to the solution of the correlation problem.

3. Artificial Intelligence Approaches

In addition to the classical and formal specification approaches used for establishing correspondences between data elements and for software component retrieval, the use of artificial intelligence techniques have been applied to the correlation problem. Two of the more promising methods investigated involve 1) the exploitation of natural language information in correlating data elements, and 2) the use of neural networks to learn the similarities among data given instances of that data.

a. Natural Language Techniques

Foremost among the natural language techniques is the use of full-text information retrieval technology to extract syntactic and semantic information needed to establish correspondence between data elements. When integrating software systems, much if not most of the information required to use formal specification approaches is not available for legacy systems. However, a good deal of syntactic and semantic information is available in the component system data dictionaries and data description languages as textual information. Full-text information retrieval technology can be used to exploit the information contained in these documents for data correlation.

(1) Full-text Information Retrieval. DELTA (Data Element Tool-Based Analysis) [BFH+95] provides a methodology to correlate data elements in existing legacy systems. The methodology involves the use of a set of prescribed tools to perform the correlation. The approach is centered on the use of full-text information

retrieval technology and exploitation of the relationships between data elements. The methodology outlines a bottom-up process whereby component system data elements are correlated to corresponding data elements in another system or to those contained in a pre-defined standard.

DELTA utilizes a step-by-step process employing a number of commercially available software tools for inputting, organizing, searching, correlating, and storing data. The first tool, a full-text information retrieval system called *Personal Librarian (PL)*, is used to match corresponding data elements in different systems. PL uses information readily available from the legacy systems' data dictionaries and database tables to compare items between systems. In addition, DELTA exploits relationships between data elements to help validate the correspondences returned by PL. The relationships between data elements are captured in the form of IDEF1X (Integration DEFinition [for information modeling] First Edition Extended) logical data models. For the task of modeling the data, DELTA uses ERwin, one of several commercial off-the-shelf (COTS) tools available for producing the IDEF1X models. DELTA also employs commonly available personal computer (PC) office automation products such as Microsoft Excel and Microsoft Word. Finally, PC Access Tool (PCAT), distributed by the DoD, provides access to data elements currently being standardized by the defense community.

The DELTA process consists of a sequence of four primary steps used to correlate data elements between systems:

1. System metadata is gathered, organized, and reformatted in order to identify the data elements to be reconciled for each system.
2. Data elements of one system are correlated with those of other systems or with standardized elements.
3. Relationships between data elements are used to validate the data element correspondences returned in the previous step.
4. A composite model of the federation data elements is created after correspondences between data elements have been validated.

Data Extraction. There are a number of sources for obtaining a system's metadata, with the data dictionary and data definition language (DDL) being two of the most common. The data dictionary should contain the data element name,

data type, length, narrative description, and sample values for each data element. Data element names and the tables in which they are found can also be easily extracted from the system's DDL. The DDL is used to define a database's conceptual schema containing the files, their attributes, and the types of those attributes comprising the logical structure of the database [Gra87]. The system's DDL may be more accurate and up-to-date than the data dictionaries because they are extracted directly from the operational system.

Source code may also provide information regarding a system's data elements. In addition, other sources for input data can include:

- Copies of major system displays or reports,
- Requirements documents,
- User manuals, and
- Database design models.

The system analyst extracts the data elements for the system from the above listed metadata source documents. After extraction, the input data is reformatted and reorganized prior to beginning the initial correlation step using the Personal Librarian (PL) software. Reformatting removes extraneous material from the data dictionary and DDL files and formats the information for input to PL. Reorganization of the data involves sorting the data elements into subject categories similar to the facets introduced by Prieto-Diaz [Pri91]. These subject categories are referred to as basic concept areas (BCAs) in DELTA. The subject categories can be ascertained either from the standard data element set being used or from review of the data element's system documentation.

Data Element Correlation. The correlation problem involves taking each data element as it is represented in one of the systems and find the corresponding data element in the standard model or in each of the other systems. The difficulty in accomplishing this task is that each system uses different words and phrases to express the same idea; thus data element names and definitions can vary between systems. The Personal Librarian (PL) tool can be used to help locate potential matches, even though the concepts are expressed differently.

DELTA's most fundamental contribution to data element correlation is the use of full-text retrieval technology in its methodology. PL's full-text retrieval technology accepts natural language queries for locating corresponding data elements. When provided a user entered query, PL provides a candidate list of corresponding elements, ranks them, and displays a snapshot of the associated metadata for each candidate, automatically "floating" those elements that most closely resemble the query input to the top of the list. Parameters used by PL for its ranking algorithm include the number of query words that match the item and how frequently each word appears in the entire full-text database being searched. Because PL accepts natural language queries, no special syntax or query language is required for query formulation. While the authors did not provide any specific guidance for query formulation, similar techniques to those used in keyword searches on most commercial search engines should enhance search effectiveness. These include using key words or phrases from the data dictionary or DDL that describe the element being matched, and first attempting the search using a specific query to minimize the number of potential matches prior to expanding the search with a more general query.

PL supports Boolean searching as well as operators that search for words that are next to or near one another to be used in queries. In addition to using a conventional query, PL provides an additional *concept search* feature. *Concept search* functions similarly to Prieto-Diaz' conceptual distance graph, returning terms similar to the query in addition to those providing a specific match [Pri91]. For example, with a concept search, entering the word *cloud* would not only return elements containing the word cloud but would also return elements containing words such as *forecast*, *wind*, *ceiling* and *temperature* that are related to clouds.

Additional features of PL allow you to define a thesaurus for commonly used search words, to see a dictionary of all the words in a database, to match words with similar spellings (fuzzy search), to use wild cards, and to reuse words from previous searches. The inclusion of a thesaurus capability is particularly useful for attempting to correlate data elements where acronyms and abbreviations are used.

Correspondence Validation. To validate the correspondences that are identified in the correlation process, a comparison must be made of the relationships between the candidate entities and other entities surrounding them in their respective data models. Only by ensuring that the entities are used in the same manner can the analyst be assured of the correspondence between two data elements. The relationships between entities are expressed in the form of an entity-relationship diagram used to model a system's data elements. DELTA uses the IDEF1X data-modeling standard to represent the entity-relationship diagram. Logic Works' ERwin tool provides the capability to create, refine, and compare data models depicted in IDEF1X. These entity relationships are then used to validate correlation assumptions and conclusions.

Composite Model Creation. After data element correspondences have been validated against the IDEF1X entity-relationship diagrams using ERwin, a composite data model for the correlated items can be manually created. Reasons for creating such a composite model include the desire to consolidate multiple databases into one or the need to identify standardized data element definitions across a number of systems. Models are first reconciled within the active system to which they belong and then to a core system (if defined). The composite model can be used to define a "standard" representation for the real-world entities involved in system interoperation. This "standard" representation can then be used as the intermediate representation in a 2-step translation process, reducing the number of required translations from $n(n-1)$ to $2n$ for a federation of n systems. The reconciled data model can also be used to submit a data element standardization package to propose additional data standards or alteration of existing standards if desired.

Measures of Effectiveness. The initial correlation process using PL is an iterative process where the system analyst submits a query for a data element, reviews the data element correspondences and then modifies the query or utilizes a different search method to locate additional elements related to the given entity. Because of the user-driven, iterative nature of the process, the authors did not provide measurements of precision and recall for the correlation process. They did provide a measure of the effort required for the overall DELTA process. A single analyst was able

to correlate approximately 200 data elements across four databases in one work week- the equivalent of approximately 5 data elements per hour or 12 minutes per attribute. Additional experimentation conducted in [CHR97] reported correlation times of approximately 15 minutes per attribute. [BFH+95]

b. Neural Networks

Li and Clifton focus on “identifying corresponding attributes in different DBMSs that reflect the same real-world class of information [LC00, p.51].” Their goal is to develop a semi-automated integration procedure that uses database metadata to extract the semantic information necessary to identify attribute correspondences. Database semantic information can be contained within the database model, within the conceptual schema, as part of application programs, or within the mind of the user. Available database metadata includes attribute names, schema design, constraints, and data contents. The authors’ approach concentrates on using the conceptual schema and database model and contents to extract the necessary semantic information; they don’t attempt to parse the application programs or “pick the user’s brains.”

The authors introduce SEMantic INTegrator (SEMINT), a neural network based tool that utilizes database metadata to identify attribute correspondences. SEMINT uses database management system (DBMS) specific parsers to extract metadata (schema design, constraints, and data content statistics) from databases to be integrated. The metadata forms a *signature* describing the attributes of the databases. The attribute signatures are used to train a neural network to recognize the signatures and thus identify corresponding attributes based on similar attribute signatures. Neural networks can learn the similarities among data directly from instances of the data and, without prior knowledge of any data relationships or patterns, empirically infer correspondences between data attributes.

The attribute correspondence process used by SEMINT involves a five-step procedure. The following overview of the semantic integration procedure is presented in Figure III-3 below.

Step 1. Metadata in the form of schema information and data content statistics is extracted from an individual database using DBMS specific parsers.

- Step 2. The metadata is normalized as a series of attribute vectors containing values of data content-based discriminators for each attribute. The attribute vectors are input to a self-organizing map algorithm that categorizes attributes into *cluster centers* according to their proximity to other attribute vectors on the map.
- Step 3. The cluster centers are used to train a three-layer backward propagation neural network to recognize attribute categories;
- Step 4. The trained neural network is provided the attribute information from another database from which it provides the similarity between each input attribute of this new database and each attribute category from the original database used to train the network.
- Step 5. System users check and confirm the similarity results returned by the trained network. [LC94, LC00]

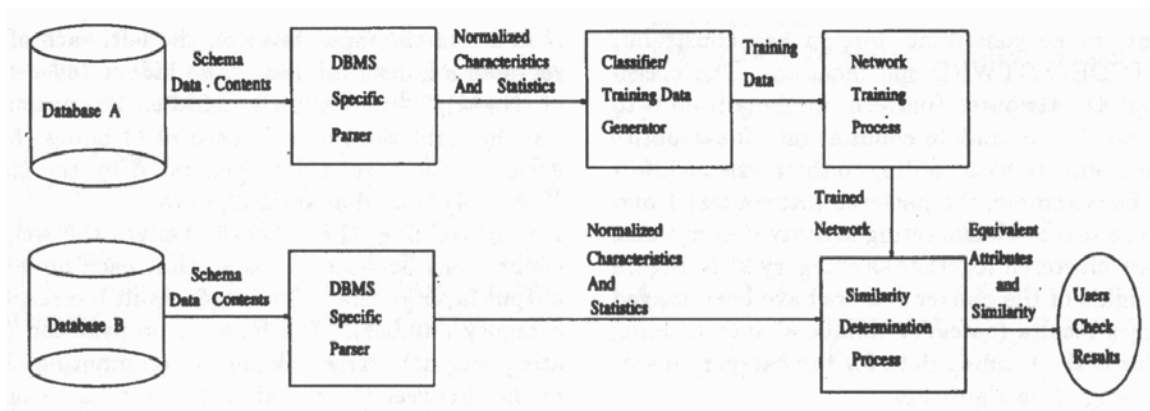


Figure III-3. Semantic Integration Procedure (From [LC94, LC00])

(1) Metadata Extraction Using DBMS-Specific Parsers. There are three levels of metadata that can be automatically extracted from a database: attribute names (dictionary level), schema information (field specification level), and data contents and statistics (data content level). Most of the past work in attribute correlation has focused on metadata available at the dictionary level. However, due to problems with the use of non-English attribute names and the use of synonyms and homonyms, approaches relying on the use of attribute names for identifying correspondences have been largely unsuccessful. SEMINT avoids the use of dictionary level metadata and focuses on

utilizing metadata at the field specification level and data content level (average number of occurrences, variation, groupings, etc.).

Field specification level information used by SEMINT includes data types, length, scale, precision, and the existence of constraints, such as primary keys, foreign keys, candidate keys, value and range constraints, disallowing null values, and access restrictions. Data contents can be used to supplement the name and schema information for determining attribute correspondences.

Data content information includes statistics on data contents such as the maximum, minimum, average, variance, coefficient of variance (square root of variance divided by the average), existence of null values, existence of decimals, scale, precision, grouping and number of segments. For numerical fields, the actual values in the fields are used to compute these statistics. For character fields, the statistics are computed based on the number of bytes used to store the data rather than the actual value of the data in the fields.

Other semantic information such as security constraints and behavior constraints such as the use of cross-references, views, clusters, sequences, synonyms, and dependencies can also be extracted from the system data dictionary. A complete list of metadata discriminators used in SEMINT is provided in Table III-1. SEMINT automatically extracts schema information and constraints from the database catalogs and computes statistics on the data contents using database queries.

(2) Normalization of Metadata. Inputs to the neural network used by Li and Clifton are required to be in the range of [0.0, 1.0], indicating whether a neuron is triggered or not [LC00]. Therefore the values assigned to the metadata discriminators of an attribute must lie in the range of [0.0, 1.0], even though the actual value of the discriminator can be of different types with different value ranges. Consequently, input values for SEMINT's neural networks must be normalized to fit within a range of [0.0, 1.0].

For each attribute, information for the various discriminators is extracted from the database, transformed into a single format, and normalized. For each database, the parser outputs a set of vectors, one for each attribute, which consists of a list

of normalized discriminator values for that attribute. These vectors represent the signature of the attribute in terms of its schema design and data content patterns.

Table III-1. Metadata and Data-Content-Based Discriminators Used in SEMINT
(From [LC00])

Table 1
Metadata (items 1–15) and data content based discriminators (items 16–20) used in SEMINT

No.	Discriminator	Descriptions
1	Data length	
2	Character type	
3	Number Type	
4	Date Type	Valid dates
5	Row ID	Data type: Row pointer
6	Raw data	Raw binary of variable length
7	Check	Constraint exists on column values
8	Primary key	
9	Unique value	Value is unique but is not part of the key
10	Foreign key constraint	Column refers to key in another table
11	Check on View	
12	Nullable	Null values allowed
13	Data Precision	
14	Data Scale	
15	Default	Has Default value
16	Minimum	Minimum non-blanks for character attributes
17	Maximum	Maximum non-blanks for character attributes
18	Average	Average non-blanks for character attributes
19	Coefficient of variance	CV of non-blanks for character attributes
20	Standard deviation	SD of non-blanks for character attributes

(3) Classifier. A classifier algorithm is used to cluster attributes from a single database into related categories. Grouping information in a database into clusters reduces the problem size and resultant neural network training time by reducing the number of nodes in the network. SEMINT uses an unsupervised learning algorithm, an adaptation of Kohonen's *Self-Organizing Map (SOM)* algorithm [Koh87], to classify attributes within a single database. The discriminator values from the attribute vector are used to plot each attribute in an N-dimension space, where N is equal to the number of discriminators used. Clusters are defined by grouping attributes whose plot falls within a predefined radius defined by the user. Cluster composition was determined from the user's input of the number of clusters to be created in Kohonen's original SOM algorithm; Clifton and Li's modification employs a user-entered cluster radius rather than the number of clusters to determine cluster composition. The classifier outputs a vector of cluster center weights for each cluster that is computed by taking the average of the discriminator values for the attribute vectors contained within a cluster.

(4) Category Learning and Recognition Neural Networks. The cluster center weight vectors output by the classifier are used as training data for a back-propagation network. Then, when the trained neural network is provided signatures of attributes from another database, it indicates if there are any matches between the clusters output from the classifier algorithm and attributes in the other database. The neural network provides a value for each cluster in the network indicating the similarity between the attribute and the target database cluster. Output values range from 1.0 for “equal” to 0.0 for completely dissimilar.

(5) Experimental Results. The authors presented their results from using SEMINT to provide attribute correspondences for three separate database integration problems. In the first problem, they used SEMINT to correlate attributes from similar databases from the same organization that contained a good deal of common information. For this application they achieved recall values of 100% and precision values between 90 and 100%. Similar results were obtained in an experiment that split one large database into two halves and then attempted to correlate the two halves, achieving recall values of close to 90% and precision values of approximately 80%. However, in a third experiment, conducted on two diverse databases containing related but not common information, results were much less favorable. In this experiment, which is closer to the application expected for establishing correspondences between component system classes in a federation, the authors achieved a recall of 38% and a precision of 20%. Additional experimentation conducted in [CHR97] reported recall values in the 20% range.

c. Applicability of Artificial Intelligence Approaches to Interoperability Correlation Problem

The two artificial intelligence based correlation approaches investigated, the exploitation of natural language information in correlating data elements, and the use of neural networks to learn the similarities among data from field specifications and data content, appear to offer the greatest potential for application to the interoperability correlation problem. Personal Librarian’s (PL’s) full-text retrieval capability is well-suited to correlating component system classes given the text-based nature of the semantic information contained in legacy system data dictionaries and data definition

languages [BFH+95]. Similarly, SEMINT's use of neural networks to determine semantic correspondence of data elements based on metadata and data element instances makes it equally suitable for use in the interoperability context [LC94]. These two approaches are complimentary- PL is based on informal descriptions and interpretations of data elements and SEMINT is based on the formal structure of data, including data types and constraints. Therefore, a combination of the two approaches should provide greater correlation success than either one alone.

C. SUMMARY

In this chapter I provided the foundation for selecting the methodology for correlating component and federation representations of the real-world entities involved in the interoperation of a federation of systems. Correlation measures of effectiveness were provided for evaluating candidate correlation approaches. Relevant existing correlation methodologies were then reviewed to provide an understanding of the various approaches and to provide an evaluation of the methodology against the prescribed correlation measures of effectiveness, where possible. Chapter VI discusses the selection of the correlation methodology for use in the OOMI IDE and provides details of the implementation of the approach used for correlating component and federation representations of the real-world entities involved in the interoperation among systems.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. OBJECT-ORIENTED METHOD FOR INTEROPERABILITY (OOMI)

A. INTRODUCTION

As stated in Chapter I, the goal of this research is to provide a computer-aided methodology to aid in the resolution of differences among independently developed heterogeneous systems in order to enable system interoperability. Interoperability was defined in Section I.A as the capability of systems to exchange information and to jointly execute tasks [LISI98, Pit97]. The information exchanged between interoperating systems consists of data used to capture the state of the real-world entities in the problem domain being modeled by systems in the federation. The joint execution of tasks reflects the capability of one system to employ the services of another. These services define the behavior of the real-world entities being modeled. Thus, interoperation can be characterized in terms of the real-world entities whose state information is exchanged between systems or whose services are invoked by another system.

The model used to portray a real-world entity can vary among independently developed systems. By their nature, two development teams charged with independently modeling the same problem domain would most likely produce different models of the real-world entities involved due to diverse perspectives of the problem, the use of equivalent constructs to model the problem environment, or dissimilar objectives for the resulting models [BLN86]. These modeling differences must be resolved if the systems are to interoperate.

B. METHODS FOR RESOLVING HETEROGENEITY AMONG SYSTEMS

Holowczak and Li provide a survey of methods for resolving heterogeneity in multidatabase systems [HL96]. The authors discuss four techniques for representing the correspondence among heterogeneous database attributes as well as the means for resolving the heterogeneity between attribute representations. The covered methodologies are:

1. The use of tables to capture attribute correspondences between heterogeneous databases;
2. The use of formulae to represent the correspondence between attributes in one database and those of another;
3. An ontological representation of attribute correspondence; and
4. A modeling approach to resolving heterogeneity issues.

In the table-based approach, corresponding attributes are listed in a row of the table, with each column in a row representing a different database representation. Additional columns in the correspondence table can be used to capture any conversions required to resolve differences between representations. Each table row forms a tuple, which represents a static correspondence between attributes in different databases.

Tables are straightforward to implement and are adequate for addressing simple attribute correspondences and for resolving naming, format, and structural heterogeneity. The major disadvantage with the use of tables is their inability to deal with higher arity attribute correspondences (e.g. one-to-many) and other forms of heterogeneity, such as heterogeneity of scope.

Attribute correspondences can also be represented by using a formula that maps attributes in one database to an attribute in another. The formula-based method is an extension of the previously discussed correspondence table and as such is capable of addressing the heterogeneity issues covered by the table-based method. In addition, functions defined by such formulas are able to resolve conflicts between atomic and composite keys as well as resolve heterogeneity resulting from missing or conflicting data. The main disadvantage of the formula-based method is its failure to assure round-tripping in the conversion between attribute representations; the inverse of a function may lose precision enabling it to only operate accurately in one direction. If the formula does not define a one-to-one function, the inverse can be multi-valued. Forcing a one-to-one relationship by choosing an arbitrary element of the set of possible inverse images is what causes loss of precision.

Another method for resolving heterogeneity in a multidatabase system is through use of an ontology to provide correspondence among database attributes. An ontology is “a knowledge base consisting of entities and relationships with abstraction, inference and typing mechanisms” [HL96, p.8]. The ontology serves as a global schema to which

attributes in component databases are associated via syntactic and semantic relationships. The associations between the global schema and component databases serve to enable translation between the component and global schema model of an attribute.

The main benefit of the ontological representation method is its ability to resolve heterogeneities of scope, level of abstraction, and temporal validity. The primary drawback to this method is related to the time and effort needed to construct the required ontology.

The modeling approach uses a high-level, typically object-oriented, model to encapsulate a component database. Heterogeneities between component databases are resolved by methods that enable translation between attribute representations. The methods may use a table, function, knowledge base, or other approach to resolve the heterogeneity issues.

An object-oriented modeling approach has the potential to address all forms of heterogeneity through the definition of custom methods. Naming differences are resolved by using attribute aliases. Format differences are handled by encapsulating the component database. Identity conflicts and missing or conflicting data can be addressed by approximating or interpolating values or by alerting the user. Structural, scope, level of abstraction, and temporal validity differences can be resolved by custom methods.

Table IV-1 compares the four methods used for resolving representations in multidatabase systems. Each method is shown with a brief description and general advantages and disadvantages of the approach.

Although the methodologies defined above have been used for mapping attribute correspondences and for resolving heterogeneity among databases, they can also be applied for achieving interoperability among a federation of independently developed systems. That is the context being investigated by the research covered in this dissertation.

Table IV-1. Comparison of Multidatabase Heterogeneity Resolution Methods
(after [HL96])

Method	Table	Formulae	Ontology	Model
Description	Attribute correspondence represented as tuples	Attribute correspondence represented as functions	Attribute correspondence represented as Articulation axioms	Attribute correspondence represented in data model
Advantages	Ease of implementation	Capable of addressing complex heterogeneity and limited inference ability	Flexible and can address complex heterogeneity. High inference ability	Flexible and can address complex heterogeneity
Disadvantages	Inability to deal with complex heterogeneity	Loss of precision problems	Requires a pre-existing robust ontology.	Limited shareability

C. OBJECT-ORIENTED METHOD FOR INTEROPERABILITY (OOMI)⁴

Upon evaluating the methods for resolving heterogeneity among multidatabase systems covered in Section IV.B against the context of the problem being investigated by this dissertation, an approach that included aspects from the formulae, ontology, and model methodologies was determined to provide the greatest benefit in resolving heterogeneities among a federation of independently developed systems. The resulting methodology, named the *Object-Oriented Method for Interoperability (OOMI)*, is based on providing a model of the real-world entities whose state and behavior are shared among systems. The methodology includes a *Federation Ontology* to provide a canonical representation for the shared information, and utilizes formulae-based methods for resolving differences between component and canonical representations of the shared information.

Due to the potential of a modeling approach to address all forms of heterogeneity, and the following benefits of Object-Oriented Analysis and Design (OOAD), it was

⁴ Portions of this material originally appeared in a paper entitled *Using an Object Oriented Model for Resolving Representational Differences between Heterogeneous Systems* [YBG02].

determined that an object-oriented modeling approach offered the greatest promise for achieving system interoperability when compared with the other approaches for heterogeneity resolution. OOAD provides principles of abstraction, information hiding, and inheritance that can be employed in the resolution of differences among independently developed heterogeneous systems [KA95, WCS00].

As the basis for achieving interoperability among systems, I define an object-oriented model that captures the shared state and behavior for a federation of systems. The resulting *Federation Interoperability Object Model (FIOM)* captures differences in modeling of the real-world entities whose state and behavior are shared among systems and provides the means for resolving such differences. Principal objectives of the FIOM are to:

- provide an abstract model of the real-world entities whose state and behavior are shared among federation systems, hiding the details of how that information is modeled on different systems, except when required for difference resolution;
- include the different component system models of the shared real-world entities with their abstract model in order to facilitate difference resolution;
- provide the means for resolving modeling differences among systems; and
- be extensible; adding new real-world entities whose state and behavior are shared among systems in the federation, or including new component system models of a real-world entity, should not affect contents or relationships in an existing model.

The FIOM presents a model of the shared state and behavior and provides a basis for achieving interoperability among systems. Construction of such a model for a large federation of systems could be a time-consuming, error-prone process. In order to reduce the time, cost, and potential for errors involved in building such a model, the use of computer aid is desired during model construction. The *OOMI Integrated Development Environment (OOMI IDE)* provides such computer aid to the interoperability engineer for constructing a FIOM for a specified federation of systems.

Finally, the ultimate objective in resolving modeling differences among independently developed heterogeneous systems is to enable system interoperation. The strategy employed for meeting this objective is to use the FIOM constructed prior to runtime for a specified federation of systems to enable runtime resolution of system heterogeneities. Resolving heterogeneities during runtime implies an automated approach that utilizes the FIOM for achieving system interoperability. The OOMI

achieves this objective through employment of one or more *OOMI translators* that act as intermediaries between federation components.

Figure IV-1 provides an overview of the components of the OOMI. Details of the FIOM are discussed in Section IV.C.1. An introduction to the OOMI IDE is provided in Section IV.C.2 with details provided in Chapter V. The OOMI translator is introduced in Section IV.C.3 and covered in detail in Chapter VII.

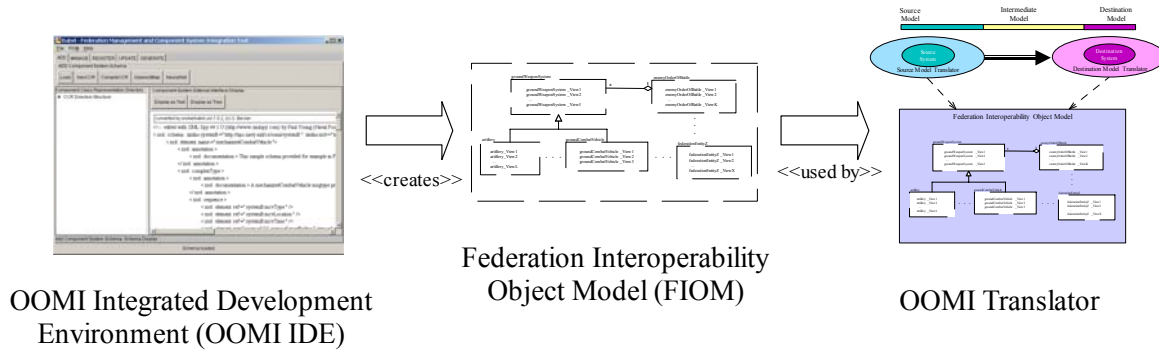


Figure IV-1. Object-Oriented Method for Interoperability (OOMI) Key Components

1. Federation Interoperability Object Model (FIOM)

Before looking at the specific components of the FIOM used for capturing the real-world entities whose state and behavior are shared between federation systems, a closer look at the types of modeling differences that can occur in heterogeneous systems presented in Section II.A.2 is provided. This examination will provide insight into the components used to model a real-world entity in the FIOM and supply a foundation for understanding the methods used for resolving heterogeneities among component system models of those entities.

a. Categories of Modeling Differences

Section II.A.2 provides a classification of modeling differences that can arise in heterogeneous systems. Further examination of this classification suggests an additional categorization of the eight classes of heterogeneity cited. Heterogeneities of scope, level of abstraction, and temporal validity all relate to differences in *what* real-world-entity characteristics are modeled by different systems. Heterogeneities of hardware and operating systems, organizational models, structure, presentation, and

meaning all relate to differences in *how* these characteristics are modeled by different systems. When two systems use different features to model the same real-world entity, then the two systems can be said to have different *views* of the real-world entity. When two systems use the same set of features to model the same real-world entity, they have the same view of the real-world entity. Two systems that have the same view of a real-world entity can nevertheless model the same feature differently. In this case, the two systems are said to provide different *representations* of the modeled feature.

To help further distinguish the difference between a *view* and the *representation* of a view, the following definition is provided. A view is defined in the OOMI as a tuple $(A\epsilon, \Omega\epsilon)$ of attribute and operation sets used to model the state and behavior, respectively, of the real-world entities involved in system interoperation. Specifically:

- $A\epsilon$ signifies the attributes $A\epsilon_1, \dots, A\epsilon_n$ contained in the model of a real-world entity that are exposed to other models in the federation.
- $\Omega\epsilon$ signifies the operations $\Omega\epsilon_1, \dots, \Omega\epsilon_n$ defined in the real-world entity model that are available for invocation by external models in the federation.

Each operation $\Omega\epsilon$ can include parameters $p_1, \dots, p_n$ used to convey the information required to perform the operation's computation or that are returned as a result of the computation.

In order for two systems to have the same view of a real-world entity, there must be no difference in scope, level of abstraction, or temporal validity between the two systems' models of the entity. In the context of the above definition, this means that at some level of aggregation each attribute set and each operation set of each system must be in one-to-one correspondence. That is, the attributes $A\epsilon$ of each system must be in one-to-one correspondence at some level of aggregation; the operations $\Omega\epsilon$ of each system must be in one-to-one correspondence at some level of aggregation; and similarly for parameters $p_1, \dots, p_n$ of corresponding operations. This does not necessarily mean that at the atomic level of attribute or operation definition that there is a one-to-one correspondence, but that such a correspondence can be provided through aggregation of one or both systems attributes and operations. For example, suppose two systems each contain an attribute that provides the geographic coordinates of a real-world entity.

System A's *position* attribute uses a latitude/longitude coordinate system and includes components *latitude* and *longitude*. System B's *location* attribute uses the MGRS coordinate system and includes components *UTM Zone*, *MGRS Northing*, and *MGRS Easting*. While there may not be a one-to-one correspondence between System A's *latitude* and *longitude* and System B's *UTM Zone*, *MGRS Northing*, and *MGRS Easting*, such a correspondence does exist between System A's *position* and System B's *location* attributes.

In addition to the requirement for a one-to-one correspondence between attribute and operation sets, corresponding operations must be *behaviorally equivalent* for two systems to have the same view of a real-world entity. In the interoperability context provided in this dissertation, behavioral equivalence of two operations is defined in terms of a black box view. Two operations are considered behaviorally equivalent if, over the complete set of possible inputs, they produce equivalent outputs from equivalent inputs when executed in the same environment. Input and output equivalence is defined in terms of the attribute and operation correspondence discussion provided in the previous paragraph. The correlation methodology discussed in Chapter VI can be used to assist the interoperability engineer in determining whether two inputs or two outputs are equivalent. Operation behavioral equivalence determination is identified as an area for future research and is left to the interoperability engineer.

To help illustrate this problem, suppose a federation of four autonomously developed military systems contained information about an enemy ground combat vehicle. From Figure IV-2 it can be seen that Systems A and B include information about the vehicle's *type*, *position*, *time*, and *range*. System C captures *type*, *position*, and *time* information on the entity, and System D utilizes *type*, *position*, *time*, and *status* to describe the vehicle. For Systems A and B there is a one-to-one correspondence between attributes $A\epsilon_{A1}$ (*type*), $A\epsilon_{A2}$ (*position*), $A\epsilon_{A3}$ (*time*), and $A\epsilon_{A4}$ (*range*) from System A and attributes $A\epsilon_{B1}$ (*type*), $A\epsilon_{B2}$ (*position*), $A\epsilon_{B3}$ (*time*), and $A\epsilon_{B4}$ (*range*) from System B. Thus, Systems A and B are said to provide the same *view* of the ground combat vehicle. Because the attributes exposed in the external interface for system C ($A\epsilon_{C1}$ (*type*), $A\epsilon_{C2}$ (*position*), and $A\epsilon_{C3}$ (*time*)) do not exhibit a one-to-one correspondence with the attributes

exposed by Systems A or B, System C is considered to provide a different view of the real-world entity than that provided by Systems A and B. Similarly, the attributes exposed by System D's model of the ground combat vehicle ($A\epsilon_{D1}$ (*type*), $A\epsilon_{D2}$ (*position*), $A\epsilon_{D3}$ (*time*), and $A\epsilon_{D4}$ (*status*)) provide a third view of the real-world entity. This example illustrates differences in view of a real-world entity among system models based on the attributes exposed in the system's external interface ($A\epsilon$); information contained in exposed operation signatures ($\Omega\epsilon$), as well as operation behavioral differences must also be evaluated in order to determine whether two models provide the same view of a real-world entity.

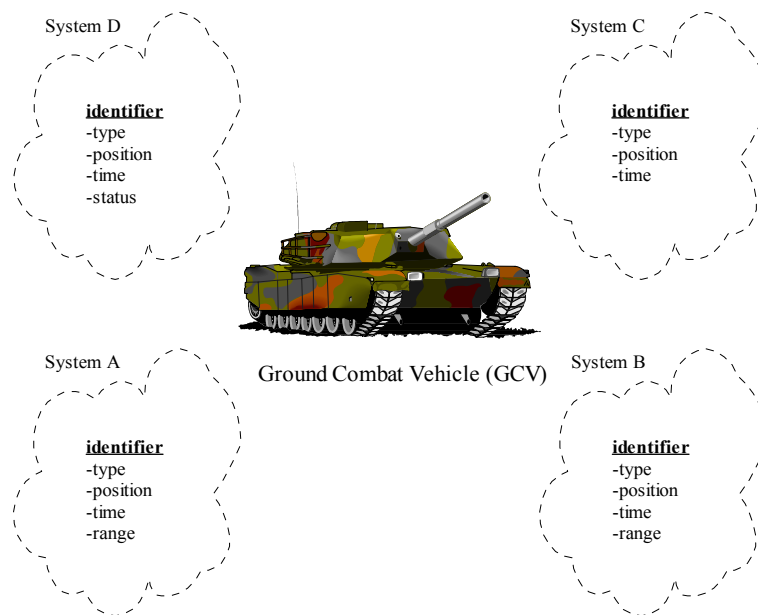


Figure IV-2. Differing Views of Real-World Entity

Even if two systems provide the same view of the entity being modeled, there may still be differences in the way their attributes, operation names, and operation parameters are represented in terms of heterogeneity of hardware and operating systems, organizational models, structure, denotation and meaning. This difference in *representation* is illustrated in Figure IV-3 by systems A and B. Even though these systems both have the same view of our real-world entity, i.e. both capture the *type*, *position*, *time*, and *range* for the entity; they each contain differences in the way that

information is represented- as a result of heterogeneities caused by hardware and operating systems, organizational models, structure, presentation, or meaning. For example, System A refers to our entity as an *Armored Combat Vehicle* and names its type attribute *acvType*. System B refers to our entity as a *Mechanized Combat Vehicle* and names its type attribute *mcvType*. Additionally, System A captures the entity's position as *acvPosition* recorded using its latitude/longitude coordinates, and the time of the vehicle data entry as *acvTime* using Greenwich Mean Time (GMT) as the reference; whereas System B records entity *mcvLocation* using Military Grid Reference System (MGRS) coordinates and records *mcvTime* using Local Mean Time (LMT). Finally, System A records the vehicle's combat range as *acvRange* measured in nautical miles (nm) whereas System B records the same quantity as *mcvRadius* measured in kilometers (km). Determining whether two models of a real-world entity that provide different representations for the entity's attributes and operations present the same view is a task that will be discussed further in Chapter VI. Figure IV-3 illustrates the different views of the example real-world entity and the various representations provided for each view.

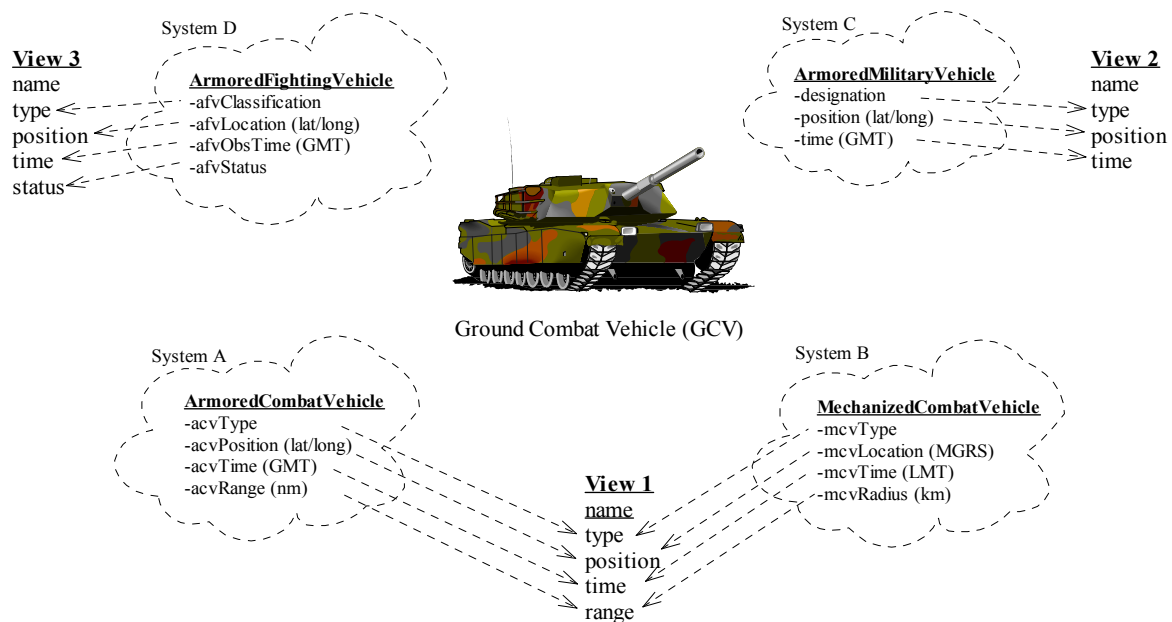


Figure IV-3. Differing Real-World Entity View Representations

b. *FIOM Composition*

It is expected that for a federation of heterogeneous systems, a number of real-world entities will be involved in the interoperation among systems. This collection of real-world entities is used to define a *Federation Interoperability Object Model (FIOM)* under the OOMI. The FIOM serves to capture 1) the real-world entities involved in system interoperation, 2) the different views a component system model might provide of these entities, 3) the different ways those views may be represented, and 4) the mechanisms used for resolving differences in view and representation seen in component system models.

(1) Capturing Real-World Entities and Views. The real-world entities whose state and behavior information are shared among a federation of interoperating systems are modeled in the OOMI as a Federation Entity (FE). The FE provides an abstract model of the information being shared while hiding the details of how that information is modeled on different systems. Each FE is composed of one or more Federation Entity Views (FEVs) used to distinguish differences in scope, level of abstraction, or temporal validity of the attributes and operations used for modeling the same real-world entity on different systems. As mentioned in Section IV.C.1.a, differences in operation behavior between systems would also result in defining different FEVs for the real-world entity being modeled. Figure IV-4 depicts the OOMI archetype for a real-world entity defining the interoperation among systems, an FE, illustrating the relationship between an FE and its constituent FEVs. As can be seen in the figure, an FE is composed of one or more FEVs, each of which provides a different view of the modeled real-world entity. Additional FEV components shown in the figure will be discussed in the next two sections.

(2) Capturing Federation Entity View (FEV) Representations. In addition to differences in what characteristics are chosen to model a real-world entity, different component systems may also represent the same characteristics differently. As discussed in Section IV.C.1.a, these differences may be due to heterogeneities of hardware and operating systems, organizational models, structure, presentation, and meaning found on the different systems. In order to capture these differences, an FEV itself includes a number of components. The first FEV component seen in Figure IV-4,

the *Federation Class Representation (FCR)*, is used to reflect the “standard” (as defined by the interoperability engineer) representation used by the federation for an entity’s view. A “standard” representation is introduced to reduce the number of translations required for resolving representational differences among systems.

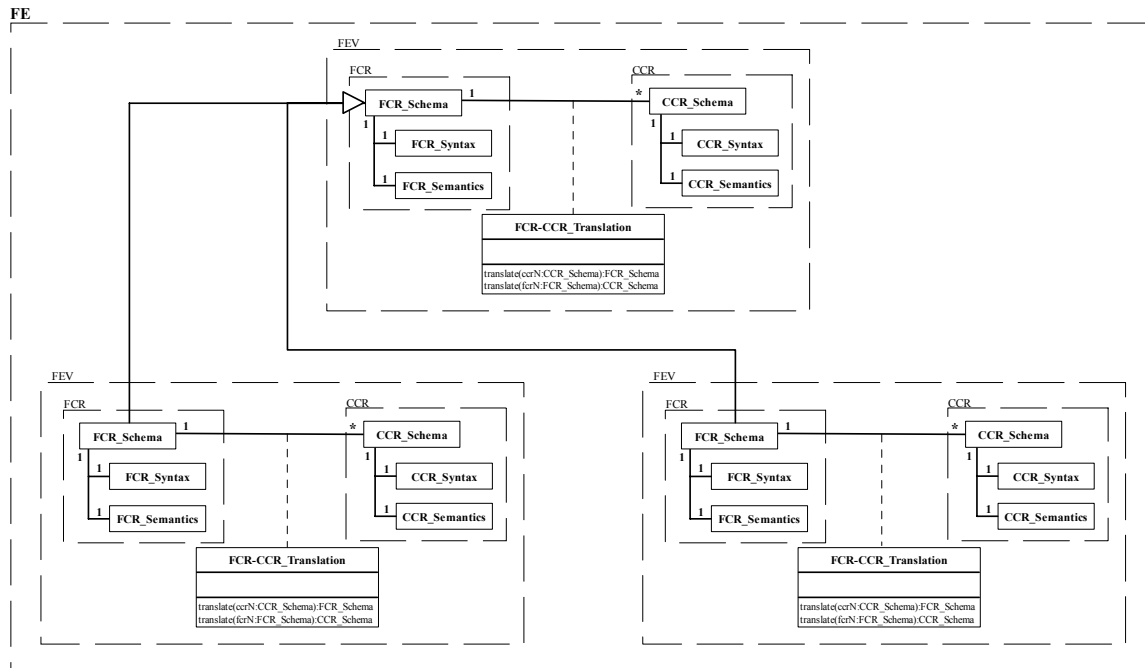


Figure IV-4. OOMI Federation Entity (FE) Archetype

The typical approach to resolving modeling differences among systems involves the use of a number of bilateral translators between systems to be integrated [RRS96]. For a federation of n systems, this approach requires the specification of $n(n-1)$ translations. An alternative to the use of bilateral translators involves the use of an intermediate representation for the real-world entities whose state and behavior information are shared among systems. Under this approach the shared information would first be converted from the source representation to the intermediate representation and then from the intermediate representation to the destination representation. The use of an intermediate representation reduces the number of required translations from $n(n-1)$ to $2n$ for a federation of n systems. The FCR provides the intermediate representation for translation between a source and destination system.

To support standardization of this intermediate representation, the terminology and representation used to define the FCR is based on an *ontology* containing the federation-sanctioned representation of an entity's state and behavior. This ontology can be developed specifically for a federation of systems or it can be derived from a domain-specific or industry-wide standard such as the Defense Information Systems Agency's (DISA's) Defense Information Infrastructure (DII) Common Operating Environment (COE) XML Registry [DII01] or the Defense Modeling and Simulation Office's (DMSO's) Functional Description of the Mission Space (FDMS) namespaces [FDM01].

The second FEV component seen in Figure IV-4, the Component Class Representation (CCR), is used to capture the possible alternative component system representations of an entity's view. The CCR defines the component system model of a real-world entity and serves as the source or destination representation for translation between models. Whereas each FEV will contain exactly one FCR that provides the "standard" representation of that view, each FEV may contain many CCR's depending on the number of component system models of a particular real-world entity view. As will be shown in Appendix A, there may also be FEV's that do not have a CCR defined for them. As depicted in Figure IV-4, an FEV contains exactly one FCR and may include zero or more CCRs.

Figure IV-5 illustrates the CCRs and FCRs created for the system A through D representation of the example ground combat vehicle previously introduced in Figure IV-3. Note that each FEV contains a single FCR whereas an FEV may contain more than one CCR- the *groundCombatVehicle_View1* FEV includes CCRs *armored-CombatVehicle_CCR* and *mechanizedCombatVehicle_CCR* corresponding to System A's and System B's representation of the view, respectively. FCR and CCR components and their relationships, to be discussed in the following paragraphs, have been omitted from the figure to enhance understandability.

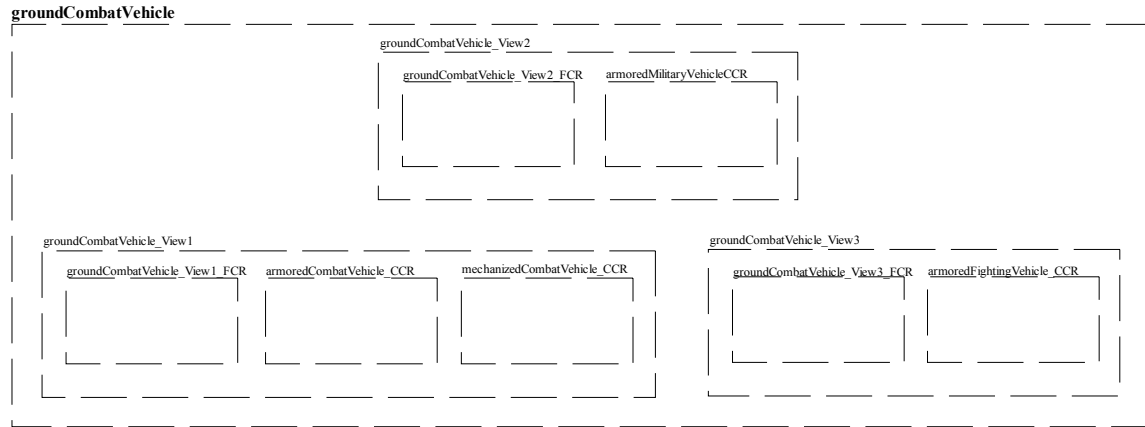


Figure IV-5. Example Views of a Federation Entity with Federation and Component Class Representations

The FCR and CCR are each actually a composition of related components. These components contain information 1) defining the attributes and operations that characterize the state and behavior of the federation or component models of a real-world entity, and 2) for identifying correspondences among models.

Identifying Federation and Component Model Attributes and Operations. As depicted in Figure IV-6, the first of these components, the *FCR Schema*, is used to capture the attributes and operations that characterize the state and behavior of a federation model of a real-world entity. In general, a schema is a summarized or diagrammatic representation of something. In our usage, the FCR Schema contains the names and type signatures of the exposed attributes ($A\epsilon$) and operations ($\Omega\epsilon$) used to provide the “standard” representation for the attributes and operations that define a particular view of a real-world entity.

Similarly, the *CCR Schema* is used to capture the attributes and operations that characterize the state and behavior of a component model of a real-world entity. The CCR Schema contains the names and type signatures of the exposed attributes ($A\epsilon$) and operations ($\Omega\epsilon$) used by a specific component system to model a real-world entity from the problem domain. The prime difference between an *FCR Schema* and a *CCR Schema* is in how they are used; the *FCR Schema* is used to characterize the state and behavior of the “standard” model of a real-world entity while the *CCR Schema*

is used to provide the same characterization for various component system models of an entity.

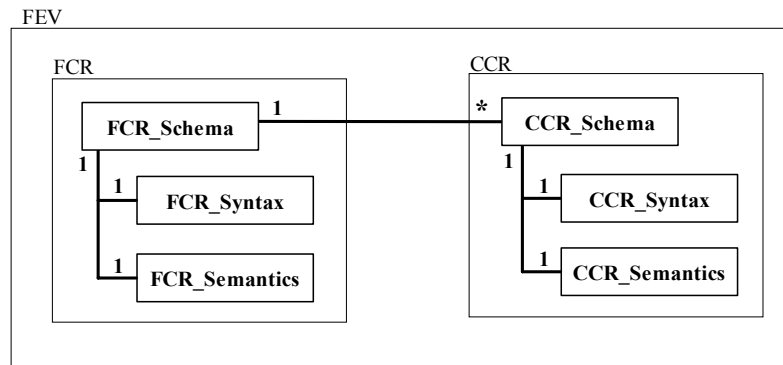


Figure IV-6. OOMI Archetype for Federation and Component Class Representations (FCR and CCR) Showing Constituent Schema, Syntax, and Semantics Classes

Whereas the FE, FEV, FCR, and CCR are conceptual constructs that may or may not have corresponding components in an FIOM implementation, it is expected that both FCR and CCR Schemas would be modeled as classes in an implementation of the FIOM. Information exchange and joint task execution among systems is effected through use of FCR and CCR Schema instances to transport information between systems.

Figure IV-7 depicts the FCR and CCR Schemas for the example groundCombatVehicle_View1 FEV from Figure IV-5. Similar FCR and CCR Schemas for groundCombatVehicle views 2 and 3 have been omitted to enhance readability. Attribute names for state information used in the FCR and CCR Schemas are taken from Figure IV-3. Operations included with the FCR and CCR Schemas are standard accessor and mutator methods for the included attributes. Additional operations exposed by federation and component models of a real-world entity would also be included in the FCR and CCR Schemas defined for an FEV. An association is established between the FCR Schema and all CCR Schemas that define the same view of the modeled real-world entity.

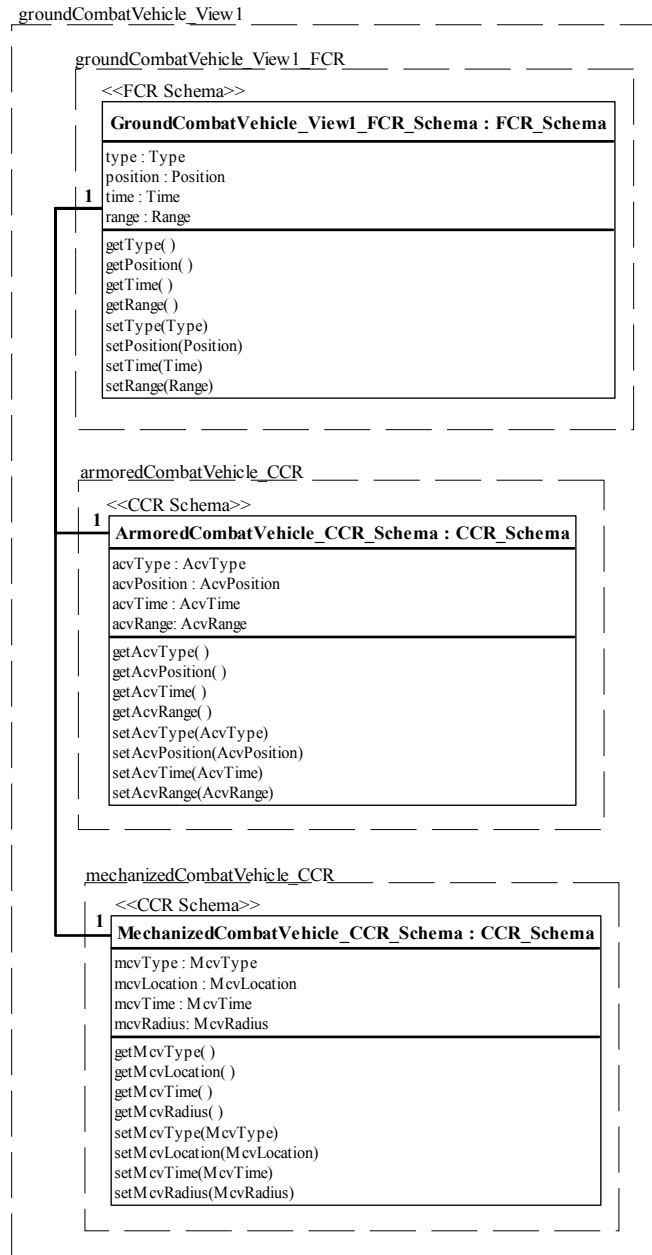


Figure IV-7. FCR and CCR Schemas for Example Ground Combat Vehicle

Identifying Correspondences Among Models. Previous efforts toward integrating heterogeneous databases found that a large part of the effort was consumed by determining whether two entries in related databases represented the same entity [LC94]. An equivalent situation exists in the integration of heterogeneous system components. When presented with a number of systems to be integrated, the interoperability engineer must determine when two systems' models of shared

information refer to the same real-world entity. Establishing this correspondence is crucial in order for systems to exchange information and operations and is the basis for defining the FEs characterizing the interoperation of federation systems.

In order to assist the interoperability engineer in establishing the correspondence between different models of a real-world entity, the FCR and CCRs also include syntactic and semantic information used to correlate the “standard” and various component system models of the real-world entities defining the interoperation. This information is represented using the components *FCR Syntax* and *CCR Syntax* to capture syntactic information on the “standard” and component representations of an FEV, respectively. Similarly, components *FCR Semantics* and *CCR Semantics* capture semantic information about the “standard” and component representations. This syntactic and semantic information is used to determine the correspondence between component system and federation models of a real-world entity in order to define the entities, views and representations of the FIOM. Once determined, this correspondence is captured in the model as an association relating an FEV’s FCR and CCR Schemas and is used later for resolving differences in representation between the models as discussed in Section IV.C.1.b(3). Figure IV-6 depicts the FCR and CCR components of an FEV, with constituent Schema, Syntax and Semantics components shown for each of them.

Syntactic information is used to capture the composition and structure of a class. Class composition is provided as a list of terms depicting the class name, and the names and type signatures of the attributes and operations contained in the class. Structural information includes data specifying the cardinality of attribute or operation occurrence, which attributes are included as parameters to which operations, whether attributes and operations are visible outside the class, etc. The composition and structure defines a *signature* for the class that can be used for comparison with other classes.

Semantics are used to provide information as to the meaning and behavior of a class, i.e., what does the state information about a class represent and what actions does the class perform? Behavioral information can be captured in terms of a narrative description of the class and its attributes and operations; a set of conditions an

operation must satisfy; or a set of equations describing the dynamic behavior of the class.

Details of the composition of FCR and CCR Syntax and Semantics classes are provided in Section VI.B.1. Their use for determining correspondence among component and federation models of a real-world entity is discussed in Section VI.B.2.

(3) Mechanisms for Resolving Differences in View and Representation. The FIOM includes two mechanisms for resolving differences in view and representation among component and federation models. The first mechanism, the *FCR-CCR Translation*, is used to resolve differences in representation between two models that have the same view of a real-world entity. The second mechanism, the *FCR Schema Inheritance Hierarchy* is used to resolve differences in view.

Resolving Representational Differences Between Component and Federation Models. Differences between component representations having the same view of a real-world entity are resolved by means of a two-step translation process whereby an instance of a source CCR Schema is first converted to an equivalent FCR Schema instance and then to a corresponding destination CCR Schema instance. As mentioned in Section IV.C.1.b(2), each FEV will contain exactly one FCR and may contain zero or more CCRs. Accordingly, an FCR may correspond to a number of CCRs in the FEV, whereas each CCR will have exactly one FCR to which it corresponds. Each CCR thus forms a unique FCR-CCR pair with its corresponding FCR. For each FCR-CCR pair, an FCR-CCR Translation is defined relating the FCR and CCR Schemas in order to enable conversion between schema instances. The FCR-CCR Translation class is expected to be implemented as an association class and will include methods used to resolve differences among models related to heterogeneities of hardware and operating systems, organizational models, structure, presentation, and meaning. The FCR-CCR Translation class is defined by the interoperability engineer and stored in the FEV for subsequent use.

Figure IV-8 illustrates the FCR-CCR Translation class used to resolve differences in representation between federation and component models having the same view of a real-world entity. An FCR-CCR Translation class contains two main methods to accomplish the conversion between schema instances. The first,

translate(ccrN : CCR_Schema) : FCR_Schema is used to convert an instance of a CCR Schema to its equivalent FCR Schema representation. The second, *translate(fcrN : FCR_Schema) : CCR_Schema* is used to convert an FCR Schema instance to an equivalent CCR Schema instance. Each of these methods contains a number of component methods used to convert between representations at an attribute-by-attribute and operation-by-operation level. These methods are of the form *ccrToFcr_fcrAttributeName(ccrAttributeName : ccrAttributeType)* to convert from a component to a federation representation of an attribute or *fcrToCcr_CcrAttributeName(fcrAttributeName : fcrAttributeType)* to convert from a federation to a component representation. Similar methods are defined for converting between operation name and operation parameter representations. Details of FCR-CCR Translation class creation can be found in Section V.D.2.c and its use in resolving representational differences during runtime OOMI translator operation in Sections VII.C.1.b and VII.C.2.c.

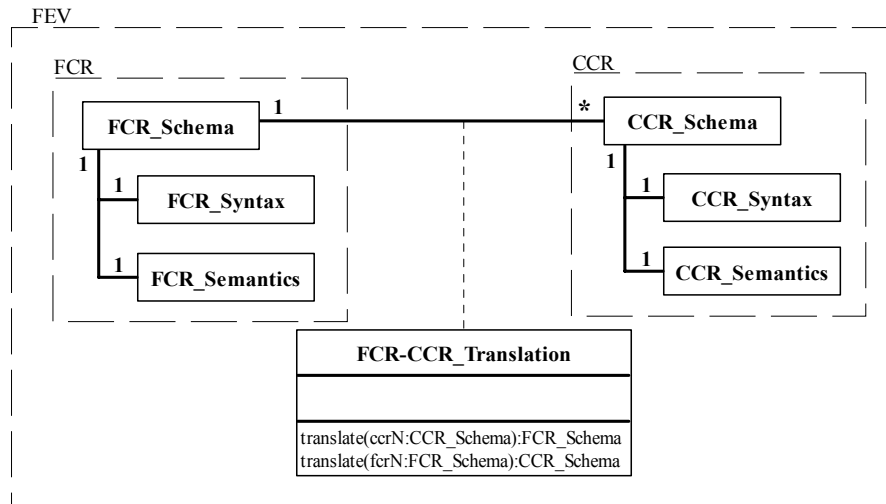


Figure IV-8. FCR-CCR Translation Class

Resolving Differences in View Among Component Models. The translations depicted in Figure IV-8 and described in the previous paragraphs enable the conversion between instances of two different schemas having the same view of the modeled real-world entity. Rarely will two different systems' view of a real-world entity

be identical. In order to share information and jointly execute tasks between two systems that have different views of the real-world entity(s) defining the interoperation, these differences in view must be resolved. Fortunately it is just as rare that different systems' views of a real-world entity are completely disjoint (otherwise they wouldn't be able to interoperate).

Generally, two or more systems' models of the same real-world entity will have some areas of commonality. Two systems' models may capture the same core state and behavior information of a real-world entity with each including additional state and behavior characteristics as required by the specific application. In this situation a schema could be defined for the core state and behavior information of the view, and separate schemas defined for the extended information. The schemas containing the extended information can be considered to be subtypes of the schema containing the common core information. Commonalities in captured state and behavior information between component system entity models enable us to determine when a supertype-subtype relationship exists between two component system schemas defining different views of the same real-world entity.

By determining the supertype-subtype relationships between component system schemas, we can construct an inheritance hierarchy that can be used to determine when the information contained in one system's view of an entity is suitable for use by another. Because there can be multiple component system schemas with the same view and since the FCR Schema provides the "standard" representation of a view, the inheritance hierarchy is constructed relating an FEV's FCR Schemas. This hierarchy is constructed by evaluating the attributes ($A\epsilon$) and operations ($\Omega\epsilon$) contained in the FCR Schema for two views. Figure IV-9 shows the FCR Schema Inheritance Hierarchy constructed for the example ground combat vehicle entity taken from Figure IV-3. Due to space considerations and to enhance understanding, containing FCRs and related CCRs with their included components are not shown with the FCR Schemas. Details of inheritance hierarchy construction and a discussion of the relationships possible between FEV schemas are provided in Chapter V and Appendix A.

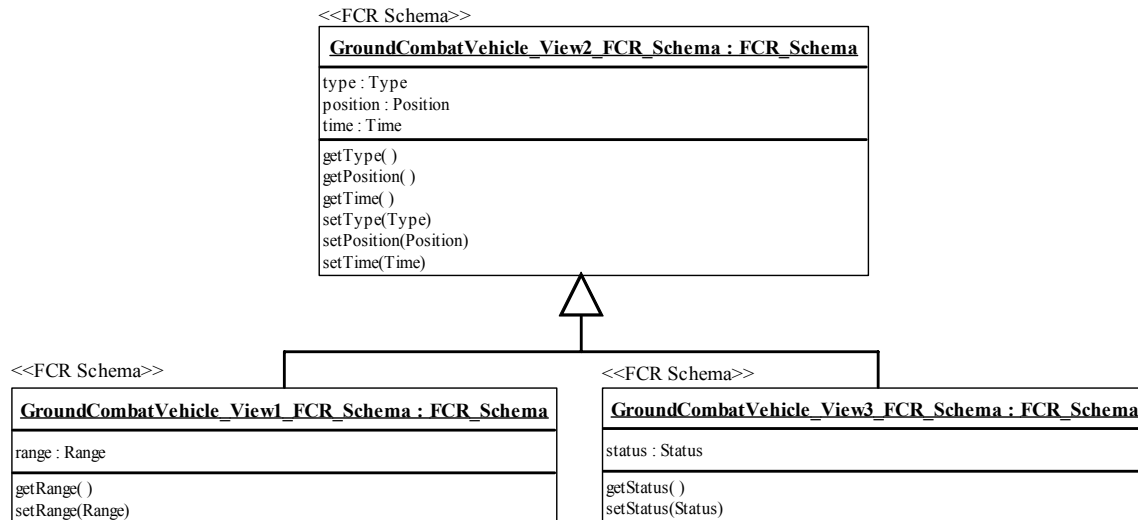


Figure IV-9. FCR Schema Inheritance Hierarchy

Determining when the information contained in one system's view of an entity is suitable for use by another is easy when the producer's view of an entity is a subtype of a consumer's view, i.e. when the producer's view schema extends the consumer's view schema. By Liskov and Wing's behavioral notion of subtyping [LW94, WO00], anywhere a supertype can be used a subtype can be substituted without any difference in behavior. Thus, in this instance the consumer will just ignore any additional information provided by the producer.

This determination is not as easy when the producer's view is a supertype of the consumer's view, or when the producer's view is not a direct ancestor or descendent of the consumer's view in the inheritance hierarchy. However, it is possible that the supertype of a real-world entity's view can be substituted for a subtype of the view under certain circumstances. These situations would include cases where 1) the attributes and operations which extend the supertype's schema are optional for the component system providing a representation of the subtype view, 2) the missing information can be obtained from the available attributes and operations via interpolation, data smoothing, etc., or 3) default values can be specified for those attributes and operations (as allowable by the destination system). Similarly, information can also be shared between component systems whose schemas are not direct ancestors or descendents of each other if there is a path in the inheritance hierarchy defined between

the producer view schema and the consumer view schema, and elements expected by the consumer that are not provided by the producer are optional or default or constructed values can be used. In order to avoid difficulties with multiple inheritance in the construction of the inheritance hierarchy, join operations where information from two different producers can be combined to create a new view satisfying a consumer's requirements are not permitted and are identified as an area for future research.

The inheritance hierarchy described above is used to resolve differences in the number and type of attributes and operations used to model a real-world entity between two systems in a federation. Differences in representation for two systems having the same view of the attributes and operations used to model a real-world entity are handled by the FCR-CCR Translation class methods included with each FEV as discussed at the beginning of this section.

2. OOMI Integrated Development Environment (IDE)

Enabling a collection of related software systems to share information and task execution has the potential for significantly enhancing the capability of the resultant federation of systems over that of the individual components. The previously introduced Object-Oriented Method for Interoperability (OOMI) is used to enable information sharing and cooperative task execution among a federation of autonomously developed systems by resolving heterogeneities among shared elements of the problem domain modeled by the federation components. In order to resolve such heterogeneities, a model of the real-world entities involved in the interoperation, termed an FIOM, is constructed for the specified system federation. Construction of the FIOM is done prior to run-time by an interoperability engineer with the assistance of a specialized toolset, the OOMI Integrated Development Environment (OOMI IDE).

The Graphical User Interface (GUI)-based OOMI IDE is used to:

1. identify the real-world entities involved in the interoperation of systems in a federation,
2. specify the different views of a real-world entity resulting from dissimilar component system perspectives of the attributes and operations required to model an entity,
3. define a "standard" federation representation for each real-world entity view identified, and establish the relationship between the "standard" view representation and various component system view representations,

4. construct an inheritance hierarchy relating the different views of a real-world entity,
5. manage a *Federation Ontology* of terms and representations used for defining the “standard” federation representation of the real-world entities whose state and behavior are shared among federation systems,
6. define the transformations required to translate between component system and “standard” representations of a view, and
7. generate system-specific information to be used by a translator to resolve modeling differences between component systems during their runtime operation.

The first task in FIOM construction is determining the real-world entities whose state and behavior are to be shared among systems in the federation. Each resultant FE is constructed from information contained in the component systems’ external interfaces or specified by an interoperability engineer. The OOMI IDE provides functionality for enabling an interoperability engineer’s input or for extracting information from a component system’s external interface definition in order to construct the FEs involved in the system interoperation. FE construction is assisted through use of a Federation Ontology containing the accepted terminology and representation to be used for defining the federation model of the real-world entity specified by an FE. The OOMI IDE provides functionality for accessing and modifying the Federation Ontology during FIOM construction.

Defining different views for an FE based on dissimilar component system perspectives of the attributes and operations required to model a real-world entity, defining a “standard” federation representation for each view, and identifying the relationships between views and between federation and component representations of a view require the interoperability engineer to identify correspondences between component and federation models of the real-world entities involved. Correlation software is used during FIOM construction for relating component and federation models of the real-world entities involved and for defining an inheritance hierarchy capturing these relationships among federation entity views and view representations.

After identifying the different representations of an FEV used by component systems, the transformations required to translate between representations must be defined. The OOMI IDE assists the interoperability engineer in this task through the use

of a GUI-based matching process used to provide computer aid to transformation development, and the maintenance of a translation library to enable the reuse of common translation algorithms.

Finally, class transformation and relationship information is extracted from the FIOM for each component system. A translator uses the system-specific information to resolve modeling differences between component systems.

Chapter V describes the functionality provided by the OOMI IDE and outlines the construction of an interoperability object model for a federation of systems, the FIOM. Correlation of component and federation real-world entity models is covered in Chapter VI.

3. OOMI Translator

Whether conducting information exchange or joint execution of tasks between systems, differences in view and representation of the real-world entities whose state and behavior are shared among systems must be resolved. The interoperability object model constructed for a specified federation of component systems during the *pre-runtime* phase is used by a *translator* at *run-time* to reconcile differences in real-world entity view and representation.

The translator serves as an intermediary between component systems. It can be implemented as part of a *software wrapper* enveloping a producer or consumer system (or both). A software wrapper is a piece of software used to alter the view provided by a component's external interface without modifying the underlying component code. Alternately, the translator could be implemented on a stand-alone system that lies between interoperating systems, such as on the hub in a hub and spoke architecture.

Figure IV-10 shows an overview of the use of software wrappers and the involvement of the FIOM in the translation process. As depicted in the figure, information or an operation signature exported by a source system is intercepted by the source model translator and converted from the source model to an intermediate model using the appropriate translation defined in the FIOM. The intermediate model of the exported information is then routed to the destination system where the destination model translator intercepts it. The destination model translator first uses the FCR Schema Inheritance Hierarchy contained in the FIOM to resolve differences in view between the

source and destination models of the exported information. The destination model translator then converts the intermediate model of the exported information to a model accepted by the destination system using the appropriate translation from the FIOM.

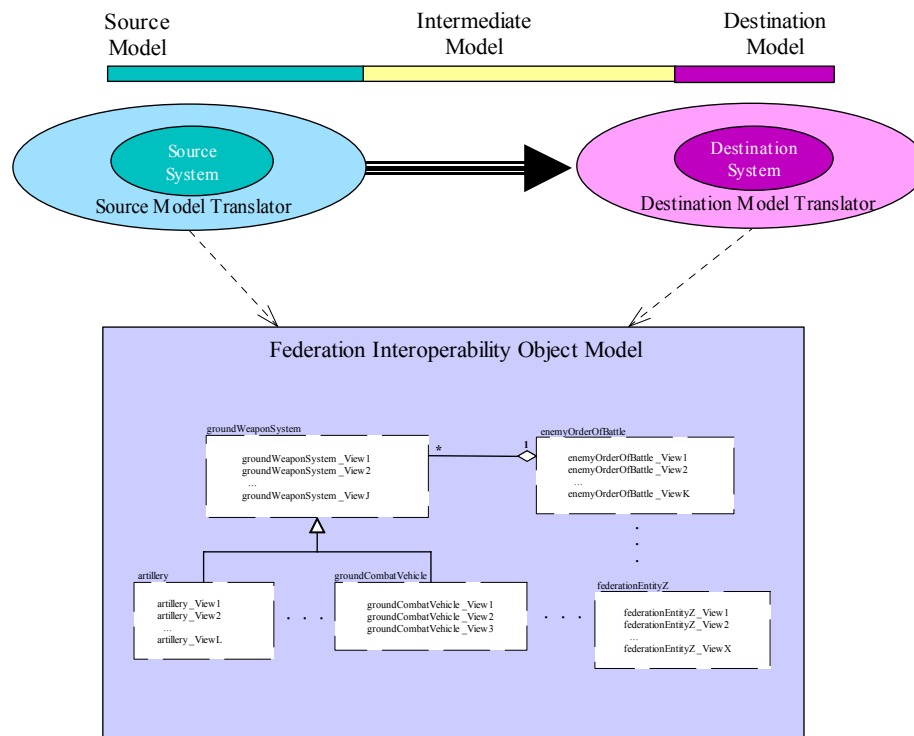


Figure IV-10. Translator - FIOM Interaction

The translations required by the translator for both information exchange and joint task execution are similar. For information exchange, the source system provides the exported information in the form of a set of attributes or objects of a producer class in the native format of the producer. In order to be utilized by a consumer system, the exported information must be converted into the format expected by the destination system. For joint task execution, a client system provides an operation name and a set of parameter values to a server system in the native format of the client. The parameters may be attributes, operations, or objects of a client class. Again, this information must be provided to the destination system in a format recognized by that system. Thus the operation name and parameter values must be converted to a form recognized by the server system in order to invoke the desired server operation.

As indicated above, the translator must be capable of converting instances of a class's attributes and operations (or both attributes and operations in the form of an object of the class) from one model to another. The information required to effect these translations is captured as part of the FIOM during federation design. As presented in Section IV.C.1.b(3), the information needed to resolve differences in what information is used to model a real-world entity between various component systems is captured in the FCR Schema Inheritance Hierarchy for each FE. Differences in how each component system represents its view of the real-world entity are resolved through the translations included as part of each FEV, illustrated in Figure IV-8. Then, at run-time, the translator accesses the information contained in the model to resolve differences in federation entity view and to effect the translation between component and standard representations of a view. Chapter VII discusses the translator in detail.

D. SUMMARY

The Object-Oriented Method for Interoperability (OOMI) introduced in this dissertation is used for resolving expected modeling differences in a federation of independently developed, heterogeneous systems in order to enable system interoperability. As the basis for achieving interoperability among systems, a model of the real-world entities whose state and behavior are shared among systems in the federation, termed a Federation Interoperability Object Model (FIOM), is defined. Disparities in component system models of these real-world entities are differentiated as to differences in *what* is modeled and differences in *how* the modeled information is represented, termed differences in *view* and *representation*, respectively.

Differences in view among models are resolved through use of an FCR Schema inheritance hierarchy relating commonalities among views provided by different component system models of the same real-world entity. Differences in representation are resolved through the use of translations to convert between models that provide the same view of the real-world entity. In addition to the FCR Schema Inheritance Hierarchy and translations used to resolve modeling differences among systems, the FIOM also includes syntactic and semantic information used during FIOM construction to establish the correspondences between different models of the same real-world entity.

Construction of an FIOM for a specified federation of systems is done prior to run-time by an interoperability engineer with the assistance of a specialized toolset, the OOMI Integrated Development Environment (OOMI IDE). The OOMI IDE provides computer aid in 1) identifying the real-world entities involved in the interoperation of systems in a federation; 2) specifying the different views and view representations a component system may provide of a real-world entity being modeled; 3) defining the “standard” federation representation of such views and representations; 4) managing the *Federation Ontology* used in federation representation definition; 5) constructing the FCR Schema inheritance hierarchies and translations needed to resolve such modeling differences; and 6) generating the system-specific information to be used by a translator to resolve modeling differences between component systems during their runtime operation.

Finally, at runtime, the OOMI translator utilizes the information contained in the FIOM to automatically resolve differences in the information exchanged between federation systems or in the operation signatures involved in joint task execution in order to enable system interoperation.

THIS PAGE INTENTIONALLY LEFT BLANK

V. OBJECT-ORIENTED METHOD FOR INTEROPERABILITY INTEGRATED DEVELOPMENT ENVIRONMENT (OOMI IDE)

A. OOMI IDE PURPOSE

As discussed in Chapter IV, resolution of modeling differences among heterogeneous components of a federation is accomplished using a two-step process. In the first step, accomplished prior to runtime, an integrated model of the real-world entities whose state and behavior are to be shared among systems in the federation, termed a Federation Interoperability Object Model (FIOM), is constructed. In the second step, performed at runtime, translator(s) reconcile view and representational differences among component systems using the FIOM.

While construction of the FIOM and its use in reconciling modeling differences in itself advances methods currently used for system interoperability, the true benefits of the Object Oriented Method for Interoperability (OOMI) lie in the foundation it provides for application of computer aid. From the discussion in Section IV.C.2, five areas are identified where computer-aid can be applied to the construction of an interoperability object model for a federation of systems. These are:

1. extraction of information contained in a component system's external interface definition to construct a federation model of the real-world entities involved in system interoperation,
2. managing the Federation Ontology used in federation model definition,
3. correlation of component and federation models of the real-world entities involved in system interoperation and construction of an inheritance hierarchy to capture the relationships between the models,
4. defining the transformations required to translate between different information representations used in the component and federation models, and
5. generating system-specific information used for runtime resolution of modeling differences among component systems.

A specialized toolset, the OOMI Integrated Development Environment (OOMI IDE) is proposed to provide computer-aid in the above areas to the development of an FIOM for a federation of component systems. The purpose of the OOMI IDE, given a definition of the external interfaces of candidate federation components, is to construct a

model of the interoperation between systems. A component system external interface specifies the information it exports or imports as well as the operations it either makes available to external entities or requires from external systems or components in order to accomplish its objectives. From the component system's external interface definition a model, the FIOM, is constructed that defines the state and behavioral information shared between component systems and provides the mechanisms for resolving differences in view and representation of that information among systems.

B. OOMI IDE DEVELOPMENT CONSIDERATIONS

Before discussing the FIOM construction process used to define the proposed OOMI IDE and the initial IDE prototype components designed to support implementation of those processes, I first take a look at some of the high-level considerations used in developing an IDE to be used to support FIOM construction.

First among these considerations is the degree to which application of computer aid is feasible. In other words, is the FIOM construction process completely automatable or is human intervention required? Section V.C discusses the FIOM construction process and identifies which phases in the process are amenable to application of computer aid. Section V.D then describes the components of an initial prototype OOMI IDE designed to add computer aid to the FIOM construction process introduced in Section V.C.

Another consideration in the development of an IDE to support FIOM construction is the availability and format of the component system external interface definitions used in constructing the FIOM. Can an IDE be developed that can construct a model of the real-world entities involved in the interoperation of a federation of systems from any component system external interface definition? Or are there certain practical limitations that must be imposed on this definition in order to support process automation? Section V.D.2.a(2) discusses the assumptions imposed on the component system external interface definition in the initial prototype OOMI IDE implementation.

An additional consideration addressed in developing a prototype OOMI IDE is the number and capabilities of the intended user(s) of the system. Is the IDE to be used by a single interoperability engineer on a single workstation, or by multiple engineers across a system network? What capabilities are interoperability engineers expected to possess? Are they expected to be knowledgeable in the inner workings of each of the

systems proposed for inclusion in a federation or in just the domain to which the proposed federation resides? As indicated in [CY01], for the initial prototype OOMI IDE implementation, the interoperability engineer is presumed to be an experienced software engineer that is knowledgeable in the proposed federation's domain namespace, i.e., he is familiar with the terminology and representations used for defining real-world entities whose state and behavior are shared among systems in the federation. In addition, the initial prototype OOMI IDE is targeted for use by a single interoperability engineer on a single workstation with provisions for future multi-engineer use across a network.

If future federation development requirements dictate a multi-user, network based IDE implementation, then consideration must be given to whether a centralized or decentralized approach is to be used for model construction and storage. In a centralized approach to model construction, creation of the Federation Entities (FEs) used to model the real-world entities involved in system interoperation and the federation representation of those entities would be centrally controlled, with component representation creation responsibility distributed among the group of participating interoperability engineers. In a decentralized approach, interoperability engineers would be free to add to or modify the FEs and their defining federation representations as long as prescribed guidelines were followed in the names and representations chosen for federation model creation. Similarly, FIOM storage could be accomplished in either a centralized or decentralized manner. In a centralized FIOM storage strategy, the information used for FIOM construction as well as the completed FIOM would be located on a centralized FIOM server with access provided to each of the connecting IDE clients. In a decentralized storage approach, this information would be divided between each of the IDE clients.

C. FEDERATION INTEROPERABILITY OBJECT MODEL (FIOM) CONSTRUCTION PROCESS

In order to provide a better understanding of where computer aid can assist the interoperability engineer in developing a model of the real-world entities involved in the interoperation of a federation of systems, I first look at defining a process for constructing an interoperability object model for a specified system federation, the FIOM. As depicted in the Figure V-1 use-case model of the candidate FIOM construction process, I propose a five-phase process for FIOM construction. These five

phases are 1) *Load Component System External Interface*, 2) *Manage Federation Entities*, 3) *Register Component Class Representation (CCR)*, 4) *Update Federation Ontology*, and 5) *Generate System-Specific Translator Information*. There is not a firm order for execution of these phases; *Load Component System External Interface*, *Manage Federation Entities* and *Update Federation Ontology* can be performed independently. However, *Load Component System External Interface* and *Manage Federation Entities* must be accomplished prior to *Register CCR*, and *Register CCR* must occur prior to *Generate System-Specific Translator Information*.

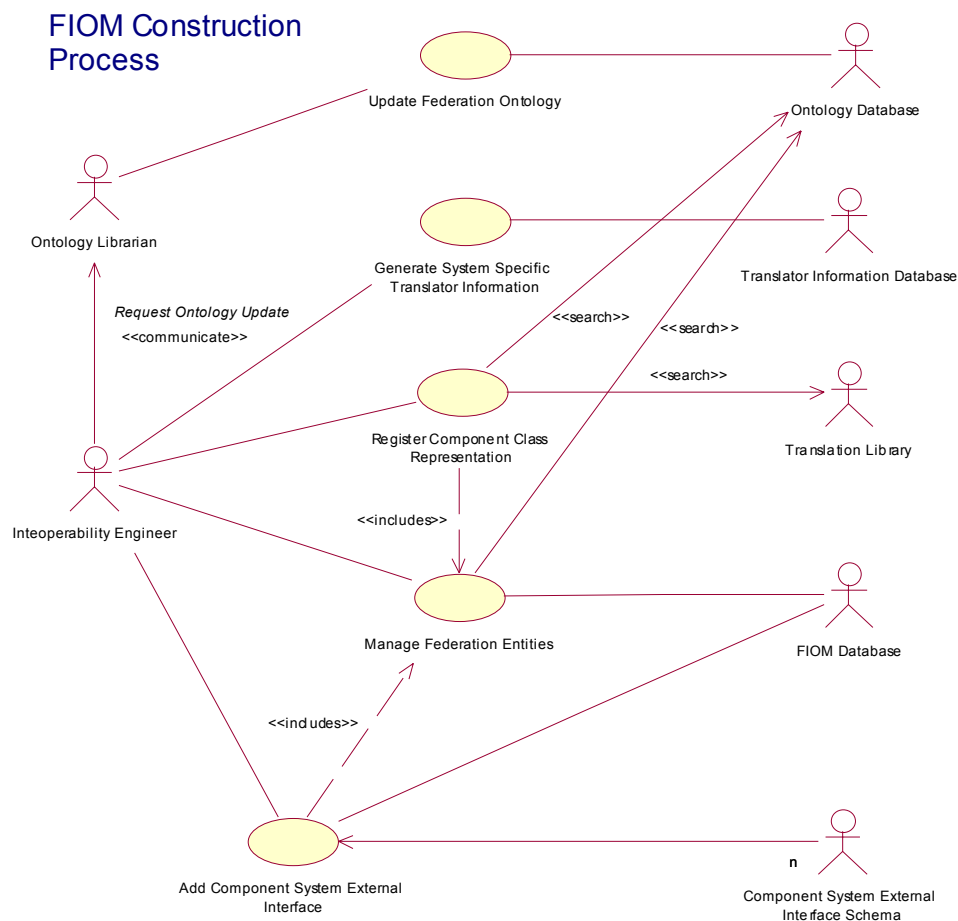


Figure V-1. Use-Case Model of Candidate FIOM Construction Process

1. Add Component System External Interface

As discussed in Section IV.C.1.b(1), the real-world entities involved in federation system interoperation are captured in the FIOM as Federation Entities (FEs). An FE includes component and federation models of these real-world entities as well as the relationships and translations required to resolve any differences between models.

A component model of a real-world entity is constructed from information contained in the component system's external interface. A component system's external interface specifies the state and behavioral information exported from or imported to the component system. From this information an interoperability engineer can identify entities in the real world to which the exported or imported information pertains. These real-world entities are modeled as classes under the OOMI. These classes, captured in the FIOM as Component Class Representations (CCRs), represent the component system's perspective of the real-world entities involved in the system interoperation. The *Add Component System External Interface* phase of FIOM development is responsible for creating CCRs from the information extracted from a component system's external interface. These CCRs are included with the FE used to represent the real-world entity in the FIOM during the *Register CCR* development phase. The process of defining a component system's CCRs from its external interface definition is one area targeted for application of computer aid in the OOMI IDE.

Definition of a component system's CCRs from its external interface specification is repeated for each component in a federation. In order to interoperate, systems must share information and operations used to model real-world entities of common interest. However, each component may present a different model of these entities making it difficult to determine that the models describe the same entity. In order to achieve interoperation among these components, the interoperability engineer must determine when two different models refer to the same real-world entity. Comparing *syntactic* and *semantic* information used to describe the models facilitates this determination. This syntactic and semantic information is extracted from a component system's external interface definition during the *Add Component System External Interface* phase of FIOM construction and saved in the CCRs defined from the external interface specification as *CCR Syntax* and *CCR Semantics* components, respectively. Details of the process used in

the OOMI IDE prototype for constructing these components are described in Section VI.B.1. Extraction of syntactic and semantic information from a component system's external interface definition and use of that information to correlate real-world entity models is another area where the OOMI IDE can provide computer aid to the interoperability engineer.

2. Manage Federation Entities (FEs)

Determining the real-world entities involved in the interoperation of systems in a federation can be done in either a top-down or bottom-up manner. The *Manage Federation Entities* phase provides the interoperability engineer with the ability to define the FEs used to characterize these real-world entities in a top-down fashion. Bottom-up FE definition is discussed in Section V.C.3. *Manage Federation Entities* provides the ability for the interoperability engineer to display the FIOM as well as modify its contents by adding or removing federation components from the model.

The ability to modify the FIOM provided in the *Manage Federation Entities* phase is designed to support the top-down definition of the FEs that comprise the interoperation among systems. During this phase the interoperability engineer can specify the FE, initial Federation Entity View (FEV), and corresponding Federation Class Representation (FCR) with defining FCR Schema for each real-world entity defining the interoperation. The terms used for defining the FE, FEV and FCR comprise an ontology used to describe the federation's representation of the real-world entities involved in the interoperation, termed the *Federation Ontology* in Chapter IV. Federation Ontology management is covered in Section V.C.4.

Along with providing the capability for the interoperability engineer to specify the contents of the FCR Schema for a federation entity, the *Manage Federation Entities* phase enables the automatic generation of syntactic and semantic information used to help correlate component representations of the entity with the federation representation. The generated syntactic and semantic information is captured as FCR Syntax and FCR Semantics components, respectively, and added to the corresponding FCR in the FIOM. Details of the syntactic and semantic generation methodology are covered in Section VI.B.1.

The interoperability engineer is also provided the ability to display the contents of the FIOM during the *Manage Federation Entities* phase. This capability includes the ability to view the FIOM as a whole or to focus on any of its FEs or their constituent components.

3. Register Component Class Representation (CCR)

The capability to display and modify the contents of the FIOM presented in the previous section is also provided during the *register CCR* phase. However, the *Manage Federation Entities* phase's focus is on defining the FE's used to characterize the real-world entities involved in the interoperation while the *Register CCR* phase is focused on adding the component model of a real-world entity to the FEV that has the same perspective of the real-world entity as the CCR being registered. Therefore, the *Register CCR* phase is concerned with 1) finding the FE and FEV that correspond to a CCR Schema being registered; 2) modifying the FIOM if necessary to provide an FE with FEV whose FCR Schema attribute and operation sets are in one-to-one correspondence with the CCR Schema attribute and operation sets; 3) adding the CCR, with constituent CCR Schema, Syntax, and Semantic components to the FEV whose FCR Schema provides the required one-to-one correspondence; and 4) adding translations necessary to resolve any representational differences between the CCR Schema being registered and its corresponding FCR Schema.

a. Finding FE Corresponding To CCR Being Registered

The first step in registering a CCR is to locate the FE in the FIOM used to model the same real-world entity as the CCR. Computer aid can be applied to this task by using the syntactic and semantic information contained in the CCR and previously stored with each FCR in the FEV. Details of the methods available for assisting the interoperability engineer in finding an FE corresponding to the CCR being registered can be found in Chapter III. Although the IDE can provide computer-aid for matching CCRs to the appropriate FE, the interoperability engineer must provide the ultimate determination of whether a CCR and FCR refer to the same real-world entity.

b. Modifying FIOM to Provide Required Correspondence Between CCR and FCR Schema

If there is no FE in the FIOM that corresponds to the same real-world entity as the CCR being registered, then the interoperability engineer must create a new FE for this real-world entity, with view corresponding to that presented by the CCR Schema. Such would be the case when adding the first component system model of a real-world entity to the FIOM. The newly created FE will include an FEV with FCR Schema containing attribute and operation sets that exhibit a one-to-one correspondence with the CCR Schema attribute and operation sets. That is, a function $f: CCR \rightarrow FCR$ must exist mapping the CCR Schema attribute and operation sets ($CCR(A\epsilon, \Omega\epsilon)$) to FCR Schema attribute and operation sets ($FCR(A\epsilon, \Omega\epsilon)$) that is one-to-one and onto and whose operations are behaviorally equivalent. New FCR Syntax and FCR Semantics components will be generated from this FCR Schema component and added to the new FEV. The interoperability engineer is responsible for defining the FCR Schema attributes and operations that correspond to the CCR Schema attribute and operation sets.

If an FE is found in the FIOM for the real-world entity modeled by the CCR being registered, then the interoperability engineer must either find an existing FEV within the FE that has the same view of the real-world entity as the CCR being registered, or must add such an FEV to the FE. The views (FEVs) of the FE are examined to determine the relationship between the Schema properties of the CCR being registered and those of each FCR defined for the FE's views.

If there is an existing FEV within the FE such that there is a one-to-one correspondence between the attribute and operation sets of the CCR Schema being registered and those of the FEV's FCR Schema, and the operation sets are behaviorally equivalent, then the CCR, with component CCR Schema, CCR Syntax, and CCR Semantics, is added to this FEV.

If the FE does not contain a view whose FCR Schema attribute and operation sets have a one-to-one correspondence with the CCR Schema attribute and operation sets or the operation sets are not behaviorally equivalent, then a new FEV with FCR Schema whose attribute and operation sets exhibit a one-to-one correspondence with the CCR Schema attribute and operation sets and whose operations are behaviorally

equivalent must be added to the FE. The new FEV will be derived from an existing view of the containing FE and its defining FCR Schema constructed either through specialization or generalization of the FCR Schema from the existing view. Details of the process for modifying the FIOM to provide an FE with FEV whose FCR Schema has the required correspondence between its attribute and operation sets and those of the CCR Schema being registered are provided in Appendix A.

c. Adding CCR to FEV Whose FCR Schema Exhibits a One-To-One Correspondence with the CCR Schema Being Registered

Once an FEV is found whose FCR Schema attribute and operation sets are in one-to-one correspondence with the CCR Schema attribute and operation sets, the CCR is added to the FEV and relationships between component and federation models of the real-world entity are established. The CCR, with component CCR Schema, CCR Syntax, and CCR Semantics, is added to this FEV and an association established between the CCR Schema and the FEV's FCR Schema. Details of the process for adding a CCR to the FEV whose FCR Schema exhibits a one-to-one correspondence with the CCR Schema being registered are provided in Appendix A.

d. Adding Translations Between Component And Federation Class Representations Of Real-World Entity

After finding or creating an FEV whose FCR Schema has the same perspective of the real-world entity as the CCR being registered and after adding the CCR to this FEV, the next task is to define the translations required to convert between the federation and component representations of that view.

The first step in defining the translations is determining whether a translation is required and whether the required translation is needed to convert from component to federation representation or from federation to component representation. A translation is required whenever source and destination representations of shared information and operations exhibit differences caused by heterogeneities of hardware and operating systems, organizational models, structure, presentation, or meaning. The basic rule in determining the direction of required translation is that if a component system provides a mechanism for exporting state information about an entity via a *send* or *return* action or for invoking an operation on another system via either a *call*, *create*, or *destroy*

action, then a translation will be required to convert from the component representation to the federation representation. Conversely, if a component system provides a mechanism for importing state information or for servicing an external operation invocation, then a translation will be required to convert from the federation representation to the component representation.

When creating translations, an interoperability engineer should be cognizant of whether there are potential consumers for information being produced by a system, or for prospective consumers whether there are systems capable of producing the desired information. Otherwise, creating a translation for unneeded or unavailable information will unnecessarily tax the resources available to the federation developer. Information regarding potential consumers or producers is available in the FCR-CCR Translation classes defined for an FE; however, because of the incremental process used in FIOM construction full knowledge of available producers and consumers would not be known prior to FIOM completion. A means for determining this information at the start of FIOM creation would be beneficial, perhaps through pre-processing of a component system's external interface definition, and is identified as an area for future research.

Once it is determined that a translation is required and the direction of translation identified, the interoperability engineer must supply the required translation. Computer aid can be applied to translation definition in two areas. First, correspondences between the attribute and operation sets contained in the FCR and CCR Schemas can be used to produce a framework for the translations required to resolve representational differences between the Schemas. Second, automated access to a library of commonly used functional translations can be provided to facilitate reuse during translation definition. Once attribute and operation mapping and functional transformation definition is completed for a translation, that translation is added to the FEV that contains the FCR and CCR for which the translation is defined, and an association established between the translation and the concerned FCR and CCR Schemas.

4. Update Federation Ontology

The OOMI includes a *Federation Ontology* to be used in defining federation model components during FIOM construction. Use of an ontology enables the standardization of terminology and representation across the federation. The ontology

provides a means for adhering to naming and presentation standardization efforts while not constraining federation terminology use when approved standards are not sufficient to meet the requirements for the desired integration effort.

During the *Update Federation Ontology* phase of FIOM development, the interoperability engineer can provide additions, deletions, and modifications to this Federation Ontology for subsequent access during FE creation in the *Manage Federation Entities* or *Register CCR* phases of FIOM development. The Federation Ontology can be produced specifically for the federation being created or it can be derived from an industry or domain specific standard and expanded or restricted as appropriate for the federation being constructed.

5. Generate System-Specific Translator Information

During the *Generate System-Specific Translator Information* phase of FIOM development, information is extracted from the FIOM for use by a specific system translator. Depending on the architecture chosen for translator implementation, a specific translator may not require all of the data contained in the FIOM to resolve modeling differences among the systems it is associated with. For example, in a wrapper-based implementation where the translator is included in a wrapper surrounding each system in the federation, the translator is only concerned with converting exported or imported information and operations between the wrapped system's model and the federation model of the real-world entities defining the shared information. Extracting only that information required for use by the specified system translator from the FIOM reduces the amount of information required by the translator to just that which is relevant to the system to which the translator is associated.

D. OOMI IDE PROTOTYPE

Based on the FIOM construction process outlined in Section V.C, a prototype Integrated Development Environment (IDE) is proposed as part of the OOMI. The prototype OOMI IDE addresses many of the development considerations called out in Section V.B and assists the interoperability engineer in creating an interoperability model for a system federation through application of computer aid to the FIOM construction process outlined in Section V.C. Major components of the prototype OOMI IDE include the: 1) User Interface; 2) FIOM Construction Manager consisting of the Federation Entity

Manager, Component Model Correlator, and Translation Generator; 3) Translator Information Generator; 4) Federation Ontology Manager; 5) FIOM Database; 6) Translation Library; 7) Federation Ontology Database; and 8) Translator Information Database. Figure V-2 illustrates the primary components of the prototype OOMI IDE.

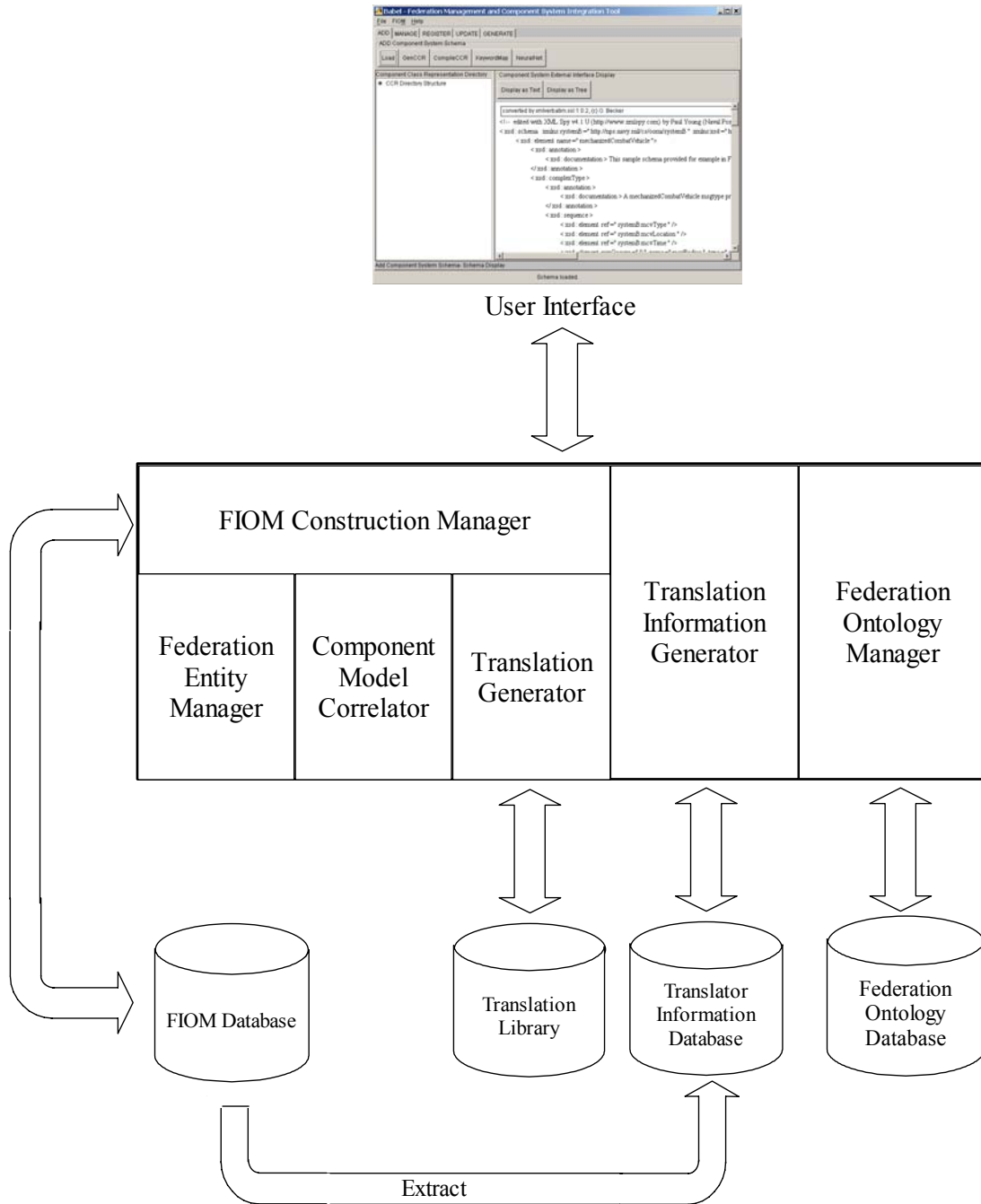


Figure V-2. OOMI IDE Block Diagram

1. User Interface

The user interface provides a GUI-based portal for the interoperability engineer to input and manipulate information required for FIOM construction, Federation Ontology management, and the extraction of system-specific information from the FIOM for use by the translator. Further detail on the OOMI IDE user interface components and display organization is provided in Section V.E

2. FIOM Construction Manager

The FIOM Construction Manager utilizes information contained in a component system's external interface definition and entered by an interoperability engineer to construct a model of the information shared among component systems of the federation. The FIOM Construction Manager includes a Federation Entity Manager, Component Model Correlator, and Translation Generator.

a. Federation Entity Manager

The Federation Entity Manager provides the capability for creating and modifying the FEs used to capture component and federation models of the real-world entities involved in the interoperation among federation systems. FE creation involves 1) initial definition of a federation model for each of the real-world entities involved in system interoperation; 2) creation of a component model of the real-world entities involved in the export or import of information and operations from the specific component systems in the federation; 3) modification of the FE and included federation model (if necessary) to include FEVs that coincide with the views presented by the various component systems' perspectives of an entity; and 4) addition of a component system's model of a real-world entity to the FEV providing the same perspective of the entity as the component model.

(1) FE Creation. FE creation can be accomplished in either a top-down or bottom-up manner. During top-down FE creation, done during the *Manage Federation Entities* phase of FIOM construction, the interoperability engineer knows up front that certain information is to be shared between federation components. This information may come from knowledge of the information and operation requirements of a component system, knowledge of the information and operations a component makes available to other systems, a desire to eliminate duplication of information or operations

by components of the federation, or specification by the federation development funding authority that certain information and operations shall be shared among systems.

The interoperability engineer specifies the FE, initial FE view (FEV), and corresponding FCR with defining FCR Schema comprising the federation model of this view from this prior knowledge. The prototype OOMI IDE enables top-down FE creation using information 1) input by the interoperability engineer; 2) contained in a specified component system's model of a real-world entity; or 3) contained in the Federation Ontology. Choice 2) above might be used for FE creation in the event that one of the component system definitions of a real-world entity were designated as the "standard" or federation representation with which other component representations must interoperate.

During bottom-up FE creation, done during the *Register CCR* phase of FIOM construction, the Federation Entity Manager uses details of the information and operations exported from or imported to a component system to identify the real-world entities whose state and behavior may be shared among systems. An object model of a component system's external interface is created for comparison with existing federation models of shared real-world entities using the Component Model Correlator discussed in Section V.D.2.b. Comparison is done to determine whether the component model corresponds to an existing federation model or identifies a new real-world entity for sharing. New entities are consequently targeted for FE creation. From this information the interoperability engineer specifies the FE, initial FEV, and corresponding FCR with defining FCR Schema for the real-world entities defining the interoperation. FCR Syntax and FCR Semantics components used in correlating component and federation models of the real-world entities involved in system interoperation are created from the FCR Schema by the Component Model Correlator discussed in Section V.D.2.b. Component model creation, discussed in Section V.D.2.a(2), must be completed prior to initiating bottom-up FE creation.

To enhance interoperability with systems added to the federation during later modifications, terminology and representations used in FE creation should be taken from the *Federation Ontology* where possible or nominated for inclusion in the

Federation Ontology if not. The terminology contained in the Federation Ontology can be derived from naming standardization efforts such as the DII COE XML Registry or FDMS namespace [DII01, FDM01]. Utilizing a sanctioned naming standard provides an added benefit should future interoperation with systems complying with such standards be required. In this case little or no modification to the FIOM and FIOM-dependent wrapper-based translators would be required. Federation Ontology management is covered in Sections V.C.4 and V.D.4.

Whether acquired by top-down or bottom-up means, the information required for FE and FCR creation is captured by the OOMI IDE as an XML Schema which will be used to automatically generate the FCR Schema defining the real-world entity's shared information and operations. The Department of Defense (DoD) is counting on XML and XML-related technologies to enable information dissemination and to resolve many interoperability issues. DoD Directive 8320.1, DoD Data Administration [DDA91], authorizes the establishment of and assigns responsibilities for DoD data administration to plan, manage, and regulate data within the Department of Defense. The Defense Information Systems Agency (DISA) is designated as the lead agency responsible for executing the policy and procedures and making DoD data standards available to the community. DISA is using XML as its common data exchange format in support of its Defense Information Infrastructure Common Operating Environment (DII COE) data engineering strategy [CY01].

The FCR Schema is automatically generated from this XML Schema through a *transformation* process. *Transformation* involves the use of XML data binding technology to convert the XML Schema into language-specific class definitions, initially targeting the Java programming language [CY01]. The FCR Schema thus created provides the federation model for the real-world entities involved in the interoperation.

(2) Component Model Creation. The next function handled by the Federation Entity Manager is the creation of a component system object model of the real-world entities whose information and operations are exported from or imported to a component system. This component system object model is created by the

interoperability engineer using Federation Entity Manager functionality and displays during bottom-up FE creation as discussed in Section V.D.2.a(1). While an object model of the interoperation among systems could be constructed from any component system external interface representation, the prototype OOMI IDE discussed in this dissertation uses XML Schema to represent the external interfaces of the component systems being integrated [ABK00].

Although the external interface for a candidate federation component might not be described in terms of the object paradigm, [GL99] indicates that defining the external interface in terms of a number of classes that represent a system's shared state and behavior is technically feasible. That, coupled with DoD's move toward adoption of XML as its common data exchange format and the ability to use XML data binding to transform the XML rendering of a component system's external interface into classes to be used for determining the real-world entities involved in the interoperation among systems, makes the assumption that a component system's external interface be defined in terms of an XML Schema reasonable [CY01].

A key pre-condition for the use of the OOMI IDE for constructing a model of the interoperation among systems in a federation is that each system's external interface is defined in terms of one or more XML Schemas, each corresponding to a real-world entity modeled by the system. The schema captures the attributes and operations defining the component system's perspective of the real-world entity as well as syntactic and semantic information providing data on the contents, structure and meaning of the entity's attributes and operations. The Federation Entity Manager provides the capability to open the XML Schema file defining the component system external interface and to create a corresponding class representation for each real-world entity defined by the XML Schema.

This class representation, captured in the FIOM as a CCR Schema, is generated from the XML Schema using XML data binding. The Federation Entity Manager uses XML data binding during the *Add Component System External Interface* phase to automatically generate a CCR Schema corresponding to the real-world entities whose information and operations are shared by the component system.

(3) Modifying FIOM (If Necessary) to Add Component Model to FE. From the CCR thus created, the interoperability engineer either locates an existing FE in the FIOM that pertains to the same real-world entity as the CCR or modifies the FIOM to add an FE for this entity during the *Register CCR* phase of FIOM development. Help in locating an existing FE is provided by the Component Model Correlator discussed in Section V.D.2.b. If during CCR registration there is no FE that corresponds to the same real-world entity as the CCR being registered, then the interoperability engineer must create a new FE for this real-world entity. The Federation Entity Manager provides the functionality for defining a new FE with initial FEV containing an FCR with defining FCR Schema corresponding to the CCR Schema being registered. The Federation Entity Manager also provides the functionality required to add the CCR to the newly created FEV.

If the FIOM contains an FE defined for the real-world entity modeled by the CCR being registered, then the interoperability engineer must either find an existing FEV within the FE whose defining FCR Schema has the same view of the real-world entity as the CCR being registered, or must add such an FEV to the FE. The Federation Entity Manager assists the interoperability engineer in adding a new FEV to the FE and in adding the CCR to the corresponding FEV. Details of FCR Schema Inheritance Hierarchy creation and modification are found in Appendix A.

(4) Adding Component Entity Model to FE. Once an FE with FEV and FCR Schema corresponding to the CCR Schema being registered is either found in the FIOM or created, the Federation Entity Manager provides the capability to add the CCR, with included CCR Schema, CCR Syntax, and CCR Semantics components to the FE. CCR Syntax and CCR Semantics components are created from the CCR Schema by the Component Model Correlator discussed in Section V.D.2.b during the *Add Component System External Interface* phase. They are used in correlating component and federation models of the real-world entities involved in system interoperation. In addition, the Federation Entity Manager adds translations, created by the Translation Generator discussed in Section V.D.2.c, to the FE during the *Register CCR* phase of FIOM development. These translations are used to resolve representational differences

between component and federation models of the real-world entities involved in system interoperation.

b. Component Model Correlator

The Component Model Correlator is responsible for establishing correspondences among information exported from or imported to component systems in a federation. These correspondences are used during the *Register CCR* phase of FIOM construction to identify the real-world entities involved in system interoperation and to create the FEs used to model a real-world entity in the FIOM. To establish these correspondences, the Component Model Correlator constructs CCR and FCR Syntax and Semantics components from the information contained in a component system's external interface description or input by the interoperability engineer. Details of CCR and FCR Syntax and Semantic component construction are provided in Chapter VI.

As discussed in Section IV.C.1.a, each component system may have a different model of the real-world entities identified in the system's external interface definition. Correlation of these models is necessary for systems interoperation. Correlation can be accomplished either by direct comparison of the component system models or by defining a "standard" model for a real-world entity and comparing each component model to the standard. As presented in Section IV.C.1.b(1), different perspectives provided by the various component system models of a real-world entity are captured in the OOMI as different views (FEVs) of an FE. For each FEV an FCR provides the "standard" representation of that view. Comparing each component representation of a real-world entity with previously defined "standard" view representations enables the interoperability engineer, with IDE assistance, to determine the relationships among component models.

The OOMI IDE correlation method involves a two-phase process, first screening FEVs on the basis of included semantic information and subsequently on the basis of the included syntactic information. Each phase produces a correlation score whereby candidate FEVs can be ranked according to best potential match. The IDE also provides the capability to choose selected FEVs from the first phase of the correlation process for submission to the second phase. The expectation is that the first phase will be used to eliminate obvious mismatches while the second phase will be used to zero-in on

potential matches. For each phase of the screening process, a threshold value can be set to minimize the number of potential matches that have to be examined by the interoperability engineer. Although the IDE provides computer-aid for matching CCRs to the appropriate FE, the interoperability engineer must provide the ultimate determination of whether a CCR and FCR refer to the same real-world entity. Details of the class correlation process are covered in Chapter VI.

c. *Translation Generator*

Given corresponding federation and component models of a real-world entity whose information and operations are shared among systems, the OOMI IDE assists the interoperability engineer during the *Register CCR* phase of FIOM construction with defining translations required to resolve representational differences between models. Computer aid is provided to the interoperability engineer in three areas. First, using correspondences between component and federation attributes and operations identified by the user, the OOMI IDE *Translation Generator* provides a framework for translation definition. Second, the OOMI IDE provides facilities for creation and maintenance of a library of pre-defined translation definitions for insertion into this translation framework. Third, the OOMI IDE provides facilities for user customization of the translations used for representational difference resolution.

The user is presented with a graphical representation of the CCR and FCR Schemas from an FEV and then given the capability to match attributes and operations between the two representations of that view via a “click-to-select” procedure. From the user’s selections a translation “skeleton” is created that maps a source attribute or operation representation to the corresponding destination attribute or operation. Differences in meaning and structural representation will automatically be resolved by the mapping procedure through the associations linking representations established by the user. Differences in presentation are resolved by the addition of functional transformation routines to the translation skeleton. These functional transformation routines can be selected from a library of previously defined common translations or created by the interoperability engineer. Once attribute and operation mapping and functional transformation definition is completed for a translation, that translation is added to the FEV containing the FCR and CCR for which the translation is defined, and

an association established between the translation and the involved FCR and CCR Schemas.

Translations between the federation and component system representations of an entity are implemented as operations of an association class between the corresponding CCR and FCR, termed an *FCR-CCR Translation* class in the OOMI. Significant consideration has been given toward the use of the eXtensible Stylesheet Language (XSL) to define the translations. The declarative nature of XSL and the availability of a number of open-source tools for use in effecting the translations made XSL appear to be the technology of choice for defining the required translations, and XSL Transformation (XSLT) the leading choice for implementation of the wrapper-based translator [ABK00, Kay00]. However, we determined that the facilities to perform other than simple functional transformations provided with the current XSLT recommendation proved inadequate for the types of functional transformations required in integrating existing legacy systems, such as the conversion of a geographic position from a latitude/longitude to a Military Grid Reference System (MGRS) representation. The ability to define such transformations using the capabilities provided by a procedural or object-oriented language enables us to handle such requirements.

Design and development of the Translation Generator for the initial prototype OOMI IDE was completed by Lee [Lee02]. The initial prototype Translation Generator enables the interoperability engineer to construct the translation framework from user-identified correspondences. Future implementations will incorporate a translation library for predefined functional transformation storage and retrieval as well as provide assistance to the interoperability engineer in identifying attribute and operation correspondences.

3. Translator Information Generator

The Translator Information Generator implements required functionality from the *Generate System-Specific Translator Information* phase of FIOM construction to extract information from the FIOM for use by a specific system translator. Depending on the architecture chosen for translator implementation, a specific translator may not require all of the data contained in the FIOM to resolve modeling differences among the systems it is associated with. This information is output to the *Translator Information Database*

discussed in Section V.D.8 for subsequent use during run-time by a specified system translator.

A *source model translator* is used to convert from the source model of the exported information or operation to the corresponding federation model of that information or operation. Therefore, if a separate translator is implemented for each component in the federation, the source model translator is only concerned with those FEs which include a CCR defined for the source system. Consequently, information extracted from the FIOM for the system's source model translator would consist of any FEs that include a CCR defined for that system. These FEs would include the translations required to resolve representational differences between the component and federation models of the exported information and operations. However, if a single source model translator is implemented for all of the components in the federation, the translator would require the information contained in all of the FE's in the FIOM.

The *destination model translator* is used to convert from the federation model of the information imported to a system or the operations requested from the system to the destination model of that information or operation request. Therefore, if a separate translator is implemented for each component in the federation, the destination model translator is only concerned with those FEs that include a CCR defined for the destination system. Consequently, information extracted from the FIOM for the system's destination model translator would consist of any FE that includes a CCR defined for that system. These FE's would include the FCR Schema Inheritance Hierarchy used to resolve differences in view between the federation model of the imported information and operation signatures and the destination system's component model as well as the translations required to resolve representational differences between corresponding views. However, if a single destination model translator is implemented for all of the components in the federation, the translator would require the information contained in all of the FE's in the FIOM.

For the initial OOMI IDE prototype, a single source and destination model translator is implemented for all of the components in the federation. Therefore the Translator Information Database includes the entire FIOM for the federation.

4. Federation Ontology Manager

The *Federation Ontology Manager* provides the capability to create and manage a federation-specific ontology of terms and representations used for defining the federation representations of the real-world entities involved in the interoperation among systems. The *Federation Ontology Manager* enables a designated member from the federation development team to create an initial Federation Ontology for the FIOM under construction and to control subsequent modifications to the ontology. The initial Federation Ontology can be constructed from an industry, organization, or domain specific ontology such as DISA's DII COE XML Registry or DMSO's FDMS, or created specifically for the FIOM under development [DII01, FDM01]. In addition, when FIOM requirements necessitate use of terminology outside of that provided by the industry, organization, or domain ontology used as the Federation Ontology baseline, the Federation Ontology Manager supports creation of a change recommendation to the baseline ontology.

Specifically, the *Federation Ontology Manager* provides the capability to:

- Search an industry, organization, or domain-specific ontology for terminology related to the federation domain,
- Import relevant information into the Federation Ontology,
- Supplement the industry, organization, or domain specific ontology with federation unique terminology, and
- Capture the federation unique terminology to support industry, organization, or domain-specific ontology change recommendation.

For the initial OOMI IDE prototype, the interoperability engineer is required to provide the terminology and representation used for elements of the federation model during FE creation. Implementation of the Federation Ontology Manager is planned for future versions of the OOMI IDE prototype.

5. FIOM Database

The OOMI IDE is used to construct an object model of the real-world entities whose information and operations are shared among systems in a federation. A separate object model, termed an FIOM under the OOMI, is constructed for each federation or federation configuration considered for interoperability using the OOMI. For each federation or federation configuration, a unique FIOM is created, each consisting of a

collection of Federation Entities (FEs), the components comprising each FE, and the relationships linking the FEs and their constituent parts. For each FIOM created or under construction, the OOMI IDE maintains a data store of the FIOM components and their relationships in order that the interoperability engineer might incrementally construct the FIOM over time or use an existing FIOM as a baseline for deriving an interoperability model for a new federation. The FIOM database provides the persistent storage mechanism for maintaining a FIOM's contents in the OOMI IDE. The Initial OOMI IDE prototype uses XML data binding technology for achieving the FIOM persistent storage capability [BOD01, Lee02].

6. Translation Library

The Translation Library provides a store of common functions for use in constructing FCR-CCR Translation class methods. The Translation Generator will provide the capability for searching the store for pertinent translation functions and for including them in the translation skeleton generated from the user-identified mapping of CCR and FCR Schema attributes and operations. The Translation Library and associated search functionality are planned for implementation in a future OOMI IDE prototype.

7. Federation Ontology Database

The Federation Ontology Database contains a compilation of the “standard” terminology and representations to be used when constructing the federation representation of the real-world entities involved in the interoperation among systems. This federation ontology should be used for defining the names, data types, field lengths, integrity constraints, etc. used in FE, FEV, FCR and FCR Schema construction. Creation and management of the database is provided by the Federation Ontology Manager discussed in Section V.D.4. The Federation Ontology Database is planned for a future OOMI IDE prototype release.

8. Translator Information Database

The Translator Information Database contains a translator-specific extract from the FIOM database created for each translator in a federation. Information contained in the database for each federation translator consists of any FE that includes a CCR defined for the system or systems that the translator interfaces with. For the initial OOMI IDE prototype, a single data store including all of the FEs contained in the FIOM is provided.

Persistent storage of the Translator Information Database is provided using XML data binding technology [BOD01, Lee02].

E. OOMI IDE PROTOTYPE USER INTERFACE DESIGN

1. OOMI IDE GUI Components

To support the FIOM construction process described in Section V.C, the OOMI IDE GUI provides the following components as depicted in Figure V-3. The *FIOM Construction Phase Folders* provide user access to the IDE functionality and information displays used to support the five phases of FIOM construction covered in Section V.C. The *OOMI IDE Toolbar* enables the interoperability engineer to select the functionality necessary for constructing an interoperability object model for a specified federation of systems. The *Directory Pane* provides a hierarchical listing of either the component system or federation model representation of the real-world entities whose state and behavior are to be shared between systems in the federation, depending on FIOM construction phase. The *Display Pane* provides either a textual or graphical representation of the component system external interfaces, a class representation of those external interfaces, or a graphical representation of the federation model, including the entities, views, and representations that comprise the model. Again, the information displayed is dependent on the FIOM construction phase.

2. FIOM Construction Phase Folders

The OOMI IDE uses a series of five tabbed folders to distinguish the functionality and display information available during the five phases of FIOM construction. The tabbed folders corresponding to the five FIOM construction phases are: ***ADD Component System External Interface***, ***MANAGE Federation Entities***, ***REGISTER Component Class Representation (CCR)***, ***UPDATE Federation Ontology***, and ***GENERATE System-Specific Translator Information***.

3. OOMI IDE Toolbar, and Directory and Display Panes

Functionality available via the OOMI IDE Toolbar and information presented via the Directory and Display Panes varies based on the FIOM construction phase. In addition, certain display information is only displayed in specific construction phases. An overview of the toolbar, directory and display panes, and other support windows contents is provided for each of the FIOM construction phase folders.

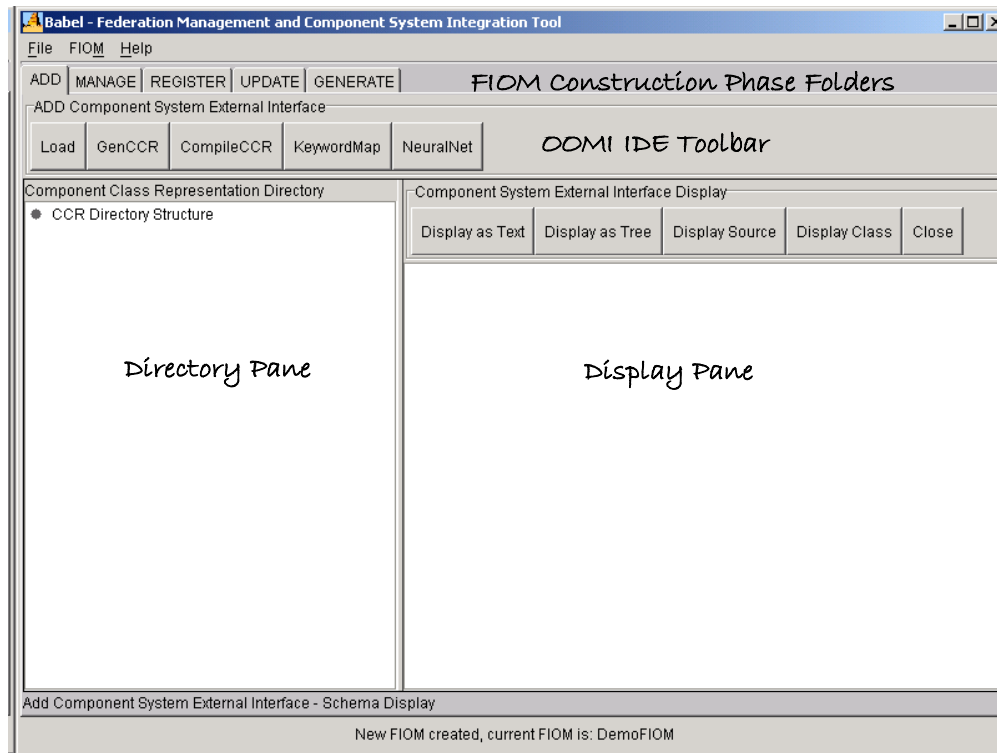


Figure V-3. OOMI IDE GUI Components

a. ADD Component System External Interface

In the first folder, *ADD Component System External Interface*, the IDE provides functionality via the OOMI IDE Toolbar to 1) load the XML Schemas that define the external interface of a component system in the federation, 2) generate a Component Class Representation (CCR) for each component model of a real-world entity defined by the XML Schema comprising the system's external interface definition, and 3) generate the syntactic and semantic information for the CCR used to correlate component and federation representations of the real-world entities involved in the interoperation. During this phase, the Directory Pane is used to provide a hierarchical listing of the CCRs generated from the component system external interface while the Display Pane is used initially to view the contents of the component system's external interface definition, and subsequently to view the content of the component classes generated from that definition. The XML Schema used to define the component system external interface can be viewed as either a textual or graphical representation.

Generated component classes can be viewed either as the programming language representation of the class or as the corresponding Unified Modeling Language (UML) representation [BRJ99]⁵. Toolbar, Directory, and Display contents for the *Add Component System External Interface* folder are illustrated in Figure V-4.

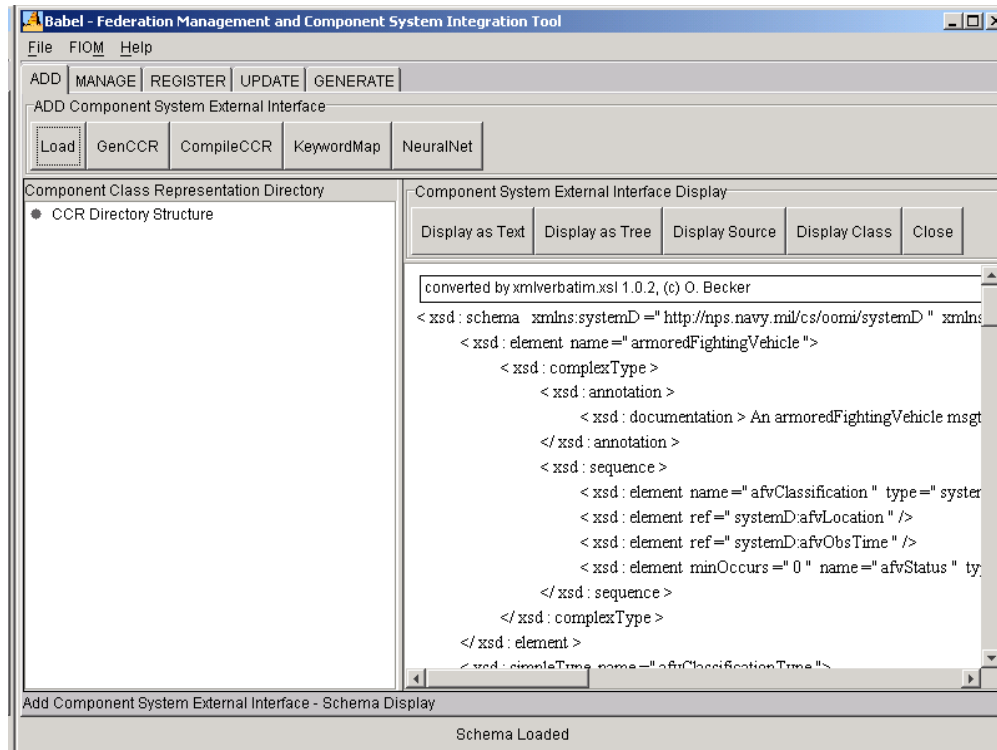


Figure V-4. *ADD Component System External Interface Folder Display and Functionality*

b. MANAGE Federation Entities

In the second folder, *MANAGE Federation Entities*, the Directory Pane provides a hierarchical listing of FIOM composition. The Display Pane provides a graphical UML depiction of the selected FIOM information⁵. The OOMI IDE Toolbar provides the capability for selecting the information to be displayed in the Display Pane as well as specifying the level of FIOM detail that the interoperability engineer wants to display. This capability provides three levels of abstraction for displaying the model of

⁵ Class models are viewed as a tree structure vice a UML diagram in the initial OOMI IDE prototype.

the federation interoperation- the *FIOM level* for an overview of the entities comprising the interoperation for the entire federation, the *Federation Entity (FE)* level for displaying a specific interoperation entity, or the *Federation Entity View (FEV)* level for seeing the details of the federation representation of a view. In addition, the OOMI IDE Toolbar provides the capability to modify the contents of the FIOM by adding or removing entities (FEs), entity views (FEVs), or Federation Class Representations (FCRs) comprising the interoperation. Figure V-5 illustrates the contents of the Toolbar, Directory, and Display for the **MANAGE Federation Entities** folder.

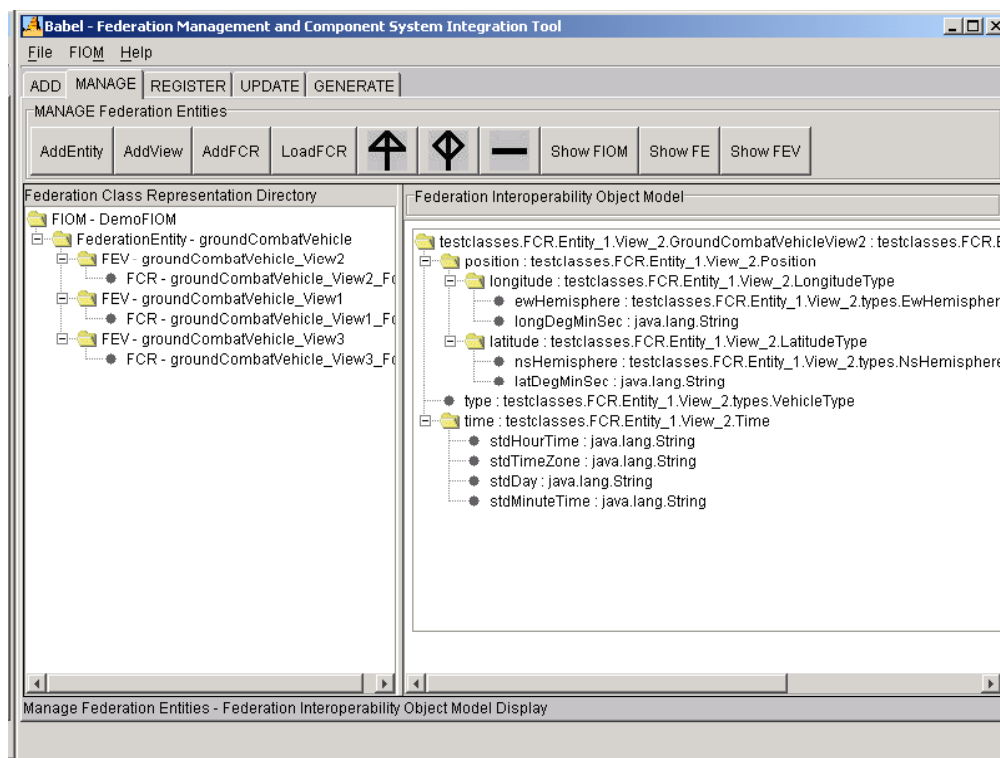


Figure V-5. *MANAGE Federation Entities Folder Display and Functionality*

c. **REGISTER Component Class Representation**

In the **REGISTER Component Class Representation (CCR)** folder the Directory Pane provides a hierarchical listing of CCRs to be added to the FIOM. The OOMI IDE Toolbar includes the functionality available in the **MANAGE Federation Entities** folder and adds the capability for adding a CCR to the FEV whose FCR Schema

attribute and operation sets exhibit a one-to-one correspondence with the CCR Schema's attribute and operation sets.

In addition to the normal Directory and Display Panes, a *Class Correlation Window* is displayed in the **REGISTER** CCR folder in order to help the interoperability engineer locate the federation view representation corresponding to a component system class representation being registered. It provides the capability to screen federation entities using the syntactic and semantic information stored with a CCR and FCR. The capability to set a threshold value for determining the display of screening results is provided for each screening phase as well as the capability to view the results of the correlation effort for verification by the interoperability engineer. Toolbar, Directory, and Display contents for the **REGISTER** CCR folder are illustrated in Figure V-6.

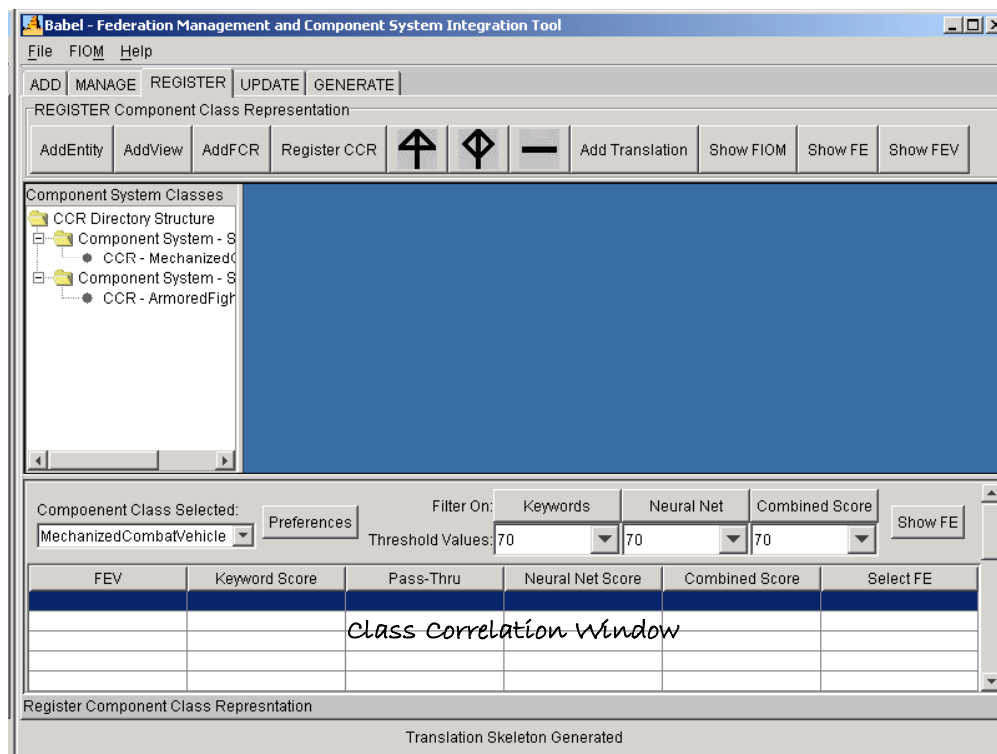


Figure V-6. *REGISTER* Component Class Representation Display and Functionality

Coincident with the capability for adding a CCR to the FEV is the functionality to define the translations required to resolve differences between the federation and component representations of the view depicted by the FEV's FCR

Schema and the CCR Schema being registered. In addition to providing the capability to display the information presented in the *MANAGE Federation Entities* folder, the Display Pane includes a *Translation Generation* window, shown in Figure V-7, that provides the capability to match attributes and operations from corresponding FCR and CCR Schemas during translation definition.

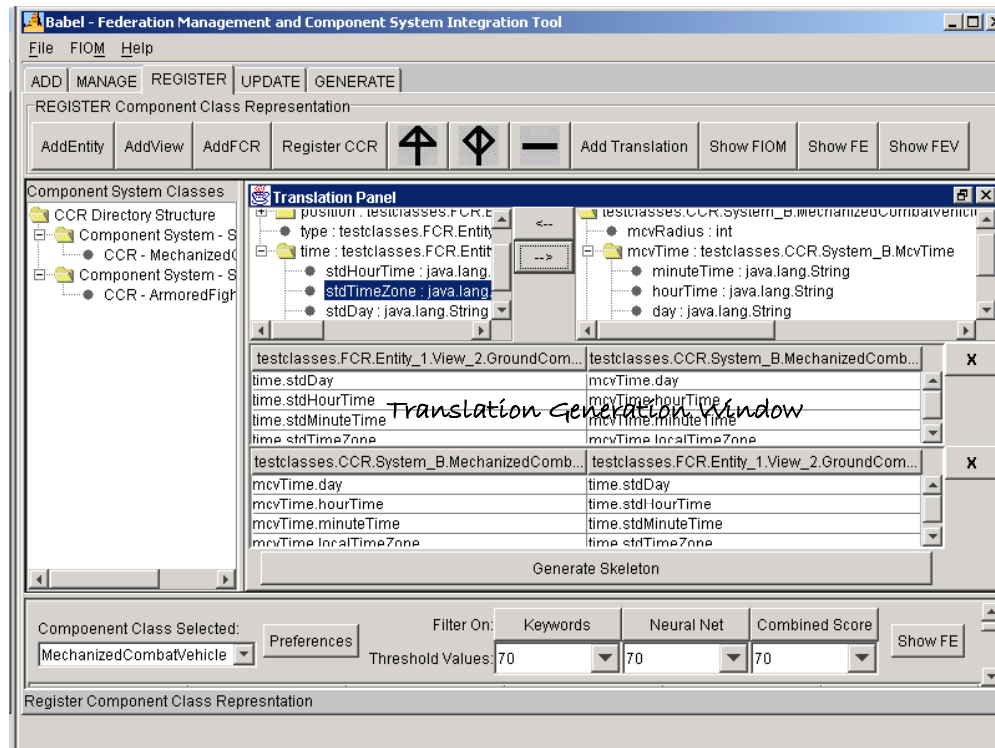


Figure V-7. REGISTER CCR Folder Translation Generation Window

From this mapping of FCR and CCR Schema attributes and operations, the Translation Generator creates a translation framework that the interoperability engineer can modify by adding functions from the Translation Library or other custom conversion methods as required. Figure V-8 depicts the resulting *translation skeleton* created from the attribute and operation mapping previously identified by the interoperability engineer.

d. UPDATE Federation Ontology

The *UPDATE Federation Ontology* folder replaces the Directory and Display panes displayed in the three prior folders with the Federation Ontology Window. The Federation Ontology window provides the capability to search a designated industry,

organization, or domain-specific ontology for terminology related to the real-world entities involved in the interoperation among systems. The Federation Ontology Window provides toolbar selections to facilitate the search and to add selected search results to the federation ontology. The toolbar also enables the ontology librarian to extend the federation ontology with program-approved terminology based on nominations from the interoperability engineer entered during the *Manage Federation Entities* or *Register CCR* phases. Finally, the **UPDATE Federation Ontology** folder OOMI IDE toolbar provides the functionality for providing a report of the terminology extensions approved by an ontology librarian for use in submitting recommendations to the appropriate naming standardization authority. Implementation of the **UPDATE Federation Ontology** folder is not included in the initial OOMI IDE prototype.

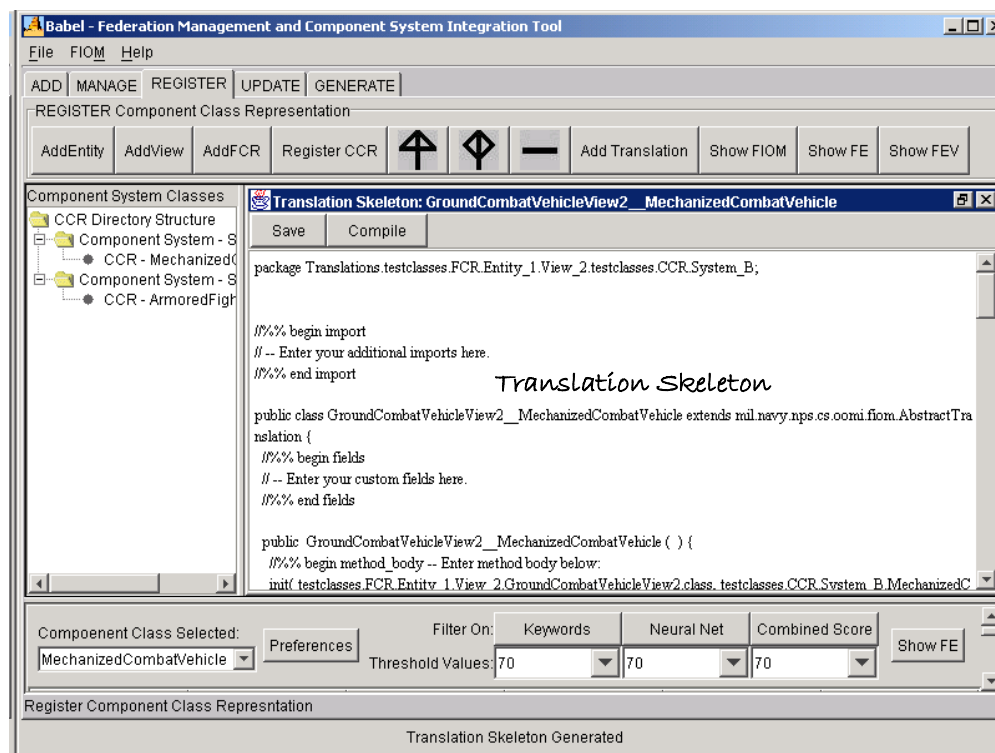


Figure V-8. Translation Generated During *REGISTER CCR Phase*

e. *GENERATE System-Specific Translator Information*

The **GENERATE System-Specific Translator Information** folder enables the interoperability engineer to extract the specific FIOM information needed by a

component system's translator. The toolbar provides functionality to select a component system for extracting system-specific translator information from the FIOM and to designate the location for storing such information for a component system translator's use. The Directory Pane provides a list of the FEs contained in a specified FIOM, enabling display of their components. The Display Pane is used for viewing the dialog boxes used by toolbar functionality. The *GENERATE System-Specific Translator Information* folder is not implemented in the initial OOMI IDE prototype.

F. SUMMARY

Under the Object-Oriented Method for Interoperability (OOMI) an integrated model of the real-world entities whose state and behavior are shared among systems in a federation, a Federation Interoperability Object Model (FIOM), is constructed prior to runtime for use in resolving differences among component system models of those real-world entities during runtime system interoperation. In this chapter, the process for constructing an FIOM is suggested and several areas where computer aid could be applied to the process identified. From the suggested FIOM construction process and areas identified for computer aid application, a top-level design for the construction of an OOMI IDE prototype is proposed. Included in the proposed top-level design, is the design for the candidate user interface implemented in an initial prototype of the OOMI IDE. Design and implementation of the prototype IDE for the OOMI were initiated in [CY01] and continued under [Lee02] and [She02].

THIS PAGE INTENTIONALLY LEFT BLANK

VI. COMPONENT SYSTEM OBJECT CORRELATION UNDER THE OBJECT ORIENTED METHOD FOR INTEROPERABILITY (OOMI)

A. CORRELATION OF COMPONENT SYSTEM AND FEDERATION REPRESENTATIONS OF A REAL-WORLD ENTITY

As discussed in the Chapter III introduction, the first step in constructing an interoperability object model for a federation of systems, the Federation Interoperability Object Model (FIOM), is determining the real-world entities that define the interoperation among systems. Identification of these real-world entities, modeled as Federation Entities (FEs) in the OOMI, can be done in either a top-down or bottom-up manner. When done top-down, the interoperability engineer uses his knowledge of information to be shared between systems to define the FEs. When defined bottom-up, the interoperability engineer must reconcile information exposed by the component systems in the federation to identify opportunities for data exchange or joint task execution. This information is used to define the FEs that delineate the interoperation among systems.

Top-down definition of FEs is fairly straightforward. The *Manage Federation Entities* capability provided in the OOMI Integrated Development Environment (IDE) enables the interoperability engineer to specify an FE for inclusion in the FIOM from information either 1) included in an XML Schema specification of the federation representation of a real-world entity, 2) contained in the Federation Ontology, or 3) input by the interoperability engineer. An XML Schema specifying the federation representation of a real-world entity might be available in the event that one of the component system definitions of a real-world entity were designated as the “standard” or federation representation with which other component representations must interoperate. Information required for FE definition could also be extracted from the Federation Ontology if sufficient information were available or input by the interoperability engineer otherwise.

Bottom-up definition of FEs is more involved. Because of potential variations in what information different component systems might view as important to model and

differences in how that information might be represented, it might not be obvious that two systems are referring to the same real-world entity. By reconciling information contained in the external interfaces of federation components, the interoperability engineer can determine when systems refer to the same real-world entities and can use this information to specify the FEs defining the interoperation among systems. This can be done in one of two ways. One option is to compare the information exposed in the external interface of all of the components in the federation in order to determine areas of commonality for use in defining the FEs. Another option is to compare the information exposed in the external interface of a component system against previously registered FEs to determine if they refer to the same real-world entity as the component system being registered, adding a new FE when a corresponding model of the real-world entity cannot be found in the FIOM.

As discussed in Section IV.C.1.b an FE encapsulates both the component and federation representations of real-world entities involved in the interoperation among a federation of systems. In order to distinguish between differences in scope, level of abstraction, or temporal validity of the attributes and operations used to model the same real-world entity on different systems, one or more Federation Entity Views (FEVs) will be defined for each FE. In addition, differences in how an FEV is represented on a specific system due to heterogeneities of hardware and operating systems, organizational models, structure, presentation, or meaning are captured as various Component Class Representations (CCRs) of that FEV. Finally the federation representation of an FEV is modeled in the FIOM as a Federation Class Representation (FCR).

Following definition of the FEs used to represent the real-world entities involved in the interoperation among systems, correspondence must be established between a component representation of a real-world entity, modeled as a CCR in the FIOM, and the federation representation, modeled as the FCR. The method chosen for bottom-up identification of FEs in the initial OOMI prototype implementation is to compare the component representation of a real-world entity against the FEs previously added to the FIOM, adding new FEs when a corresponding model of the real-world entity cannot be found in the FIOM.

The correlation of component system and federation representations of a real-world entity is the focus of this chapter. A detailed discussion of the correlation methodologies incorporation in the OOMI IDE is presented.

B. OOMI CORRELATION METHODOLOGY

The correlation methodology selected for the OOMI IDE is an adaptation of existing research in the information retrieval and software reuse communities to the problem of determining correspondence between different representations of the same real-world entity. The principle goal of the OOMI correlation methodology is to provide a phased approach that results in increasing precision in class correlation while maintaining a high level of recall. Thus, a multi-level approach was chosen for correlation of component and federation representations in the OOMI IDE.

The approach first exploits semantic information found in textual descriptions of the component and federation representations of a real-world entity. A keyword matching technique similar to that used in Personal Librarian's (PL's) full-text retrieval capability [BFH+95] is used to compare component system descriptive information with corresponding information maintained for the federation representation of the real-world entities involved in the interoperation. This first phase is designed to eliminate federation representations that are obviously not related to the component representation being registered from further consideration by the computationally more expensive second phase. The IDE provides two mechanisms for down-selecting the list of candidates for consideration by the second phase. First, the IDE enables the interoperability engineer to set a semantic correlation threshold value based on the percentage of keywords matched between component and federation representations for display of candidate matches. Second, the interoperability engineer can selectively choose from the list of federation representations whose percentage of keywords matched is above the threshold value to pass-through to the second phase. Details of the semantic matching process are contained in Section VI.B.2.a.

The second phase of the correlation effort uses a neural network based approach similar to that used by Li and Clifton in SEMINT [LC94] to explore structural similarities between component and federation representations to determine their correspondence. Li and Clifton's use of database content information in SEMINT provided a limited seman-

tic discrimination capability. Using the database content, they were able to distinguish between two elements that had the same structural characteristics but whose actual values indicated that they were not related. For example, by using data content statistics one might be able to distinguish between an employee ID and a transaction timestamp, even though both items could use a six-digit integer to represent them. The neural network based approach used in the OOMI IDE is focused on the use of field specification level information for determining syntactical correspondence between representations. However, additional data content level information such as may be available with pattern limitations, minValue, maxValue type constraints, or through run-time collection may be exploited to provide a semantic correlation capability as well. Sections VI.B.1.b and VI.B.2.b provide details of the OOMI IDE syntactic correlation process.

1. Generating Syntactic and Semantic Information Used in the Correlation Process

The first step in the correlation process is the generation of the syntactic and semantic information needed for component and federation model correlation. As introduced in Chapter V, this information is generated for each CCR during the *Add Component System External Interface* phase of the FIOM construction process. Equivalent information is generated for an FCR as FEs are added to the FIOM during either the *Manage Federation Entities* or *Register CCR* phases.

a. Generating Components Used By Semantic Matching Process

As discussed in Section VI.B, the OOMI semantic matching process uses a keyword matching technique similar to that used in PL's full-text retrieval capability. In order to correlate component and federation representations of a real-world entity, the semantic matching algorithm requires a list of keywords used by the component system and the federation to describe a real-world entity. Keyword information for a component system representation of a real-world entity is included in the XML Schema(s) used to characterize the component system's external interface. For the federation representation of a real-world entity, this information can be obtained from an XML Schema representation of that information, extracted from the Federation Ontology, or directly entered by the interoperability engineer. Table VI-1 provides a list of the fields defined in the XML schema from which keyword descriptions can be obtained [Pug01].

Table VI-1. XML Schema Fields Used for Keyword Determination (From [Pug01])

Field	Attribute	Details
xsd:element	“name”	The name attribute typically equates to the field name used in the underlying database.
xsd:element	“type”	For schemas using global types. This attribute’s value is usually descriptive of the kind of data in the subtype. (e.g. “date_type”)
xsd:documentation	N/A	The text in this element is the “description” field from the data dictionary. It is typically a human-readable free text explanation of the field’s use or format.
xsd:attribute	“name”	Gives amplifying information about a simple or complex type.
xsd:enumeration	“value”	Used to constrain the values of types. Usually used to limit a message field to several values which will reveal the use of the message (e.g.. “SUB”, “SURF”, “AIR”)

Keyword information for the component representation of a real-world entity is obtained from the XML Schema used to define a component system’s external interface. Keyword information is not extracted directly from the XML Schema, but instead it is obtained from the *CCR Schema* created from this information during the *Add Component System External Interface* phase of FIOM construction. Keyword information is extracted from the CCR Schema and stored with a CCR as a *CCR Semantics* component.

Keyword information for the federation representation of a real-world entity is obtained from an XML Schema used to describe the real-world entity, from information describing the entity input by the interoperability engineer taken from the Federation Ontology, from information specified by the interoperability engineer, or from a combination of the three sources. Keyword information is not extracted directly from these sources for use in the semantic correlation process. Instead, it is obtained from the *FCR Schema* created from this information during the *Manage Federation Entities* or *Register CCR* phases of FIOM construction. Keyword information is extracted from the FCR Schema and stored with an FCR as an *FCR Semantics* component. Use of CCR and FCR Schema for constructing CCR and FCR Semantics components provides a consistent source for required keyword information and eliminates potential difficulties

encountered during construction of CCR and FCR Semantics components when adding a new view to a Federation Entity during FIOM construction.

b. Generating Components Used By Syntactic Matching Process

As was seen with the semantic matching process, component system external interface XML schemas are built from the component system data dictionaries and data definition language. In addition to the information used by the semantic matching process, these XML schemas contain added data used by the syntactic matching process. This includes information regarding schema structure, data element type, frequency of occurrence, data size specifications, and data value constraints. This syntactic information is also included in the generated *CCR and FCR Schemas* as was the case with the semantic information.

The additional syntactic information contained in the *CCR* or *FCR Schema* component is used to support a neural network based syntactic matching process. The schema information required to support the syntactic matching process is captured in a *CCR* or *FCR Syntax* component that is added to the FIOM directory. The neural network based process uses two different subcomponents to support the correlation effort. The first subcomponent, the *CCR* or *FCR Syntax Vector*, is associated with both *CCR* and *FCR Syntax* components and contains a number of *discriminator vectors* used to capture the data content and structure information from a *CCR* or *FCR Schema*. The second subcomponent, the *FCR Syntax Net*, is associated only with an *FCR Syntax* component and contains a trained neural network for an FCR that is used in the correlation process.

A *discriminator vector* is an array of values in the range [0.0, 1.0] used to represent the data content and structure of each attribute and operation in the *CCR* or *FCR Schema*. Information used to construct the *discriminator vector* is extracted from the *CCR* and *FCR Schema* as was done in constructing the *CCR* and *FCR Semantics* components in Section VI.B.1.a. The *discriminator vectors* are added to the *CCR* or *FCR Syntax Vector* subcomponent, as appropriate.

The *discriminator vectors* are used by a neural network based matching technique to correlate component and federation representations of a real-world entity. *Discriminator vectors* in each *FCR Syntax Vector subcomponent* are used to train a neural network for the associated *FCR Schema* for later comparison with *CCR Schemas*

to determine if the FCR and CCR correspond to the same real-world entity in the problem domain. A *discriminator vector* from the *CCR Syntax Vector* subcomponent corresponding to an attribute or operation in the *CCR Schema* is provided as input to the trained neural network and an evaluation of the similarity between the *CCR Schema* attribute or operation and each attribute and operation in the *FCR Schema* is conducted.

The process for creating a *discriminator vector* is similar for both a CCR and FCR. The first step is to create a *CCR* or *FCR Syntax component* with included *Syntax Vector* subcomponent and add it to the FIOM directory. The *CCR* or *FCR Syntax Vector subcomponent* contains a *discriminator vector* for each attribute and operation in the *CCR* or *FCR Schema*. The discriminators used in creating a *discriminator vector* differ between those used for creating a schema attribute vector and those used for creating a schema operation vector.

Each component of the CCR or FCR Schema is first evaluated to determine whether it is an attribute or operation. Data content and structure information used by the OOMI IDE for creating an attribute's discriminator vector includes information concerning:

- data element structure,
- data element type,
- frequency of occurrence,
- data size specifications, and
- data value constraints

For attributes, the first data element structure discriminator, *isComplex*, distinguishes whether an attribute is complex or atomic. If the attribute is complex (i.e. the attribute is itself an object), discriminators *numSubtypes*, *numReqdSubtypes*, and *numOptSubtypes* count the total, required, and optional number of subtypes defined for that attribute, respectively. In addition, discriminator *numOperations* counts the total number of operations defined for a complex attribute and its subtypes; *numParameters* similarly counts the total number of parameters required by all the operations defined for a complex attribute and its subtypes.

For atomic attributes, type specification discriminators indicate whether an attribute is of type *string*, *boolean*, *float*, *int*, etc. For complex attributes the type

specification discriminators provide a sum of the number of attributes of each type that are defined for the complex attribute and all of its subtypes.

Frequency of occurrence discriminators *minOccurs* and *maxOccurs* specify the minimum and maximum times an attribute may be included in a real-world entity model. Data size specifications *minLength* and *maxLength* specify the minimum and maximum length of an atomic attribute of type string, whereas for complex attributes these discriminators specify the sum of the *minLengths* or *maxLengths* of all string subtypes. Discriminators *totalDigits* and *fractionDigits* provide a count of the total digits and fractional digits for attributes of type *bigDecimal*. For complex attributes these values indicate the sum of the *totalDigits* or *fractionDigits* for all *bigDecimal* subtypes.

The first data value constraint discriminator, *pattern*, indicates whether a restriction has been placed on the values allowed for string or numeric types. For atomic attributes the discriminator is Boolean; for complex attributes the *pattern* discriminator provides a count of the number of attributes among its subtypes that have a *pattern* defined. Data value constraint *numEnumerations* provides a count of the number of enumeration values specified for an atomic attribute or the sum of the number of enumeration values of a complex attribute's subtypes. Finally, *minExclusive*, *maxExclusive*, *minInclusive*, and *maxInclusive* discriminators specify a lower and upper bound to the values allowed for numeric types.

Information contained in an operation's discriminator vector is limited to a total count of the number of parameters required for operation invocation, and a count of the number of parameters by type. For CCR or FCR Schemas whose operation parameters are taken from its list of attributes, including additional values in the operation's discriminator vector would result in duplication of the information already contained in the attribute discriminator vectors without providing any additional support for discrimination among FCRs. Future correlator enhancements may include additional operation parameter discriminators for those operation parameters that are distinct from a CCR or FCR Schema's attributes. Table VI-2 provides a list of the discriminators that comprise a discriminator vector for each attribute or operation.

Table VI-2. Metadata Based Discriminators Used in Syntactic Correlation Process
(After [She02])

Number	Discriminator	Description
Structural Information		
1	propertyType	Operation or Attribute
2	isComplex	Describes whether an attribute is complex or atomic
3	numSubtypes	If attribute is complex, number of subtypes
4	numReqdSubtypes	If attribute is complex, number of required subtypes
5	numOptSubtypes	If attribute is complex, number of optional subtypes
6	numOperations	For Complex Attribute – total no. of operations defined for type
7	numParameters	For Operation – number of parameters For Complex Attribute – Sum of parameters for all operations defined for that attribute and any subtypes
Type Specifications		
8	string type	java.lang.String type
9	boolean type	primitive Boolean type
10	float type	primitive float type
11	double type	primitive double type
12	bigDecimal type	java.math.BigDecimal type
13	int type	primitive int type
14	long type	primitive short type
15	short type	primitive short type
16	other type	type other than listed above
Frequency of Occurrence		
17	minOccurs	minimum number of times attribute must occur in class modeling real-world entity
18	maxOccurs	maximum number of times attribute may occur in class modeling real-world entity
Data Size Specification		
19	minLength	For Atomic String Type Attribute – minimum length of string For Complex Attribute– Sum of minLengths of all string subtypes
20	maxLength	For Atomic String Type Attribute – maximum length of string For Complex Attribute – Sum of maxLengths for all string subtypes
21	totalDigits	For Atomic bigDecimal Type Attribute – total number of digits included in attribute For Complex Attribute – Sum of total number of digits for all bigDecimal types
22	fractionDigits	For Atomic bigDecimal Type Attribute – number of digits in fraction part of attribute For Complex Attribute – Sum of total number of digits in fraction part for all bigDecimal types
Data Value Constraints		
23	pattern	For Atomic Attribute – restriction to values allowed for string and numeric types For Complex Attribute – number of attributes with pattern defined
24	numEnumerations	For Atomic Attribute – number of enumeration values for attribute For Complex Attribute – sum of number of enumeration values for all subtypes
25	minExclusive	lower open bound of interval defined for numeric attribute types
26	maxExclusive	upper open bound of interval defined for numeric attribute types
27	minInclusive	lower closed bound of interval defined for numeric attribute types
28	maxInclusive	upper closed bound of interval defined for numeric attribute types

Neural network input values for the OOMI IDE syntactic correlator are required to be in the range [0.0, 1.0], signifying whether a neuron is triggered or not. In order to satisfy this requirement, an algorithm must be provided mapping the metadata provided for each attribute parameter to a discriminator value in the range [0.0, 1.0].

For some discriminators, such as *minOccurs* and *pattern* (for atomic attributes), the Boolean nature of the discriminator enables a direct map to a 0 or 1 value. Other parameters, such as *numSubtypes*, *numReqdSubtypes*, *numOptSubtypes*, *numOperations*, *numParameters*, *maxOccurs*, *minLength*, *maxLength*, *totalDigits*, *fractionDigits*, *numEnumerations*, *minExclusive*, *maxExclusive*, *minInclusive*, and *maxInclusive* must be normalized to a value in the range of [0.0, 1.0]. For these values Li and Clifton used a SIGMOID-like function in order to avoid false matches or false drops that could occur when using a linear normalization function. For positive numeric values in the range of [0.0, 100.0] Li and Clifton used the function

$$f(x) = 2 * (1/(1 + k^{-x}) - 0.5) \quad \text{with } k = 1.01 \quad [\text{Eq 6.1}]$$

where x is the parameter value to be normalized [LC00]. For other numeric values that may be positive or negative, they used the function

$$f(x) = 1/(1 + k^{-x}) \quad [\text{Eq 6.2}]$$

with $k = 1.01$ for values in the range of [-50.0, 50.0]. As each of the parameters *numSubtypes*, *numReqdSubtypes*, *numOptSubtypes*, *numOperations*, *numParameters*, *maxOccurs*, *minLength*, and *numEnumerations* are expected to have values in the range [0.0, 100.0], we have adapted the function from Eq 6.1 for normalizing these parameters. As parameters *maxLength*, *totalDigits*, and *fractionDigits* may have values that exceed 100 for complex attributes, a third function

$$f(x) = \log(x + 1)/5 \quad [\text{Eq 6.3}]$$

has been defined which provides adequate discrimination between values in the range of [0, 1000000]. Similarly, while parameters *minExclusive*, *maxExclusive*, *minInclusive*, and *maxInclusive* may have either positive or negative values, suggesting the use of Eq 6.2

for normalizing their values, the potential to exceed the bounds of [-50.0, 50.0] prescribed for use of that equation has led to the use of Eq 6.3 for normalizing these parameters as well.

Finally, for parameters such as an atomic attribute's data element type, the parameter is mapped to a vector of values, each in the range [0.0, 1.0]. This is done in cases where the value assigned two parameters might incorrectly convey similarity between the parameters. For example, for atomic data types, assigning a *string* parameter a value of 0.1, a *boolean* a value of 0.2, a *float* a value of 0.3, etc. would imply that an attribute of type *string* is more like a *boolean* than a *float*. However, if we instead assign each possibility a vector of values where each value in the vector is either 0.0 or 1.0, then we eliminate any incorrectly perceived "closeness" between values. For example, if we represented a string by the vector <0.0, 0.0, 1.0>, a boolean by the vector <0.0, 1.0, 0.0> and a float by the vector <1.0, 0.0, 0.0> then each would stimulate its own neuron in the neural network and there would be no incorrect perceptions as to the relationships between values. Table VI-3 and Table VI-4 provide a list of the discriminators used in the OOMI IDE and the normalized value added to the attribute or operation parameter discriminator vector for the listed discriminator value.

In addition to the *CCR or FCR Syntax Vector* subcomponents created for the CCR and FCR, a trained neural network is created for each federation entity FCR. The OOMI IDE uses a back-propagating neural net that is trained using supervised learning until it maps all neuron stimuli values to the desired outputs within a given threshold. The threshold value is set by the interoperability engineer using the OOMI IDE GUI.

Table VI-3. Discriminator Values Used for Syntactic Correlation (After [She02])

Number	Discriminator	Value to Vector
Structural Information		
1	propertyType	Operation – 0.0 Attribute – 1.0
2	isComplex	If yes – 1.0 Otherwise – 0.0
3	numSubtypes	For operation or atomic attribute – 0.0 For complex attribute - Value normalized to [0.0, 1.0] ^{Note 1}
4	numReqdSubtypes	For operation or atomic attribute – 0.0 For complex attribute - Value normalized to [0.0, 1.0] ^{Note 1} (subtype required unless minOccurs = 0)
5	numOptSubtypes	For operation or atomic attribute – 0.0 For complex attribute - Value normalized to [0.0, 1.0] ^{Note 1} (subtype optional only if minOccurs = 0)
6	numOperations	For operation or atomic attribute – 0.0 For complex attribute - Value normalized to [0.0, 1.0] ^{Note 1}
7	numParameters	For atomic attribute – 0.0 For operation – Value for # of parameters normalized to [0.0, 1.0] ^{Note 1} For complex attribute – Value for sum of parameters for all operations in subtype normalized to [0.0, 1.0] ^{Note 1}
Type Specifications		
8	string type	If atomic attribute – < 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0 > If complex attribute or operation – Value normalized to [0.0, 1.0] ^{Note 1}
9	boolean type	If atomic attribute – < 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0] ^{Note 1}
10	float type	If atomic attribute – < 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0] ^{Note 1}
11	double type	If atomic attribute – < 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0] ^{Note 1}
12	bigDecimal type	If atomic attribute – < 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0] ^{Note 1}
13	int type	If atomic attribute – < 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0] ^{Note 1}
14	long type	If atomic attribute – < 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0] ^{Note 1}
15	short type	If atomic attribute – < 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0] ^{Note 1}
16	other type	If atomic attribute – < 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 > If complex attribute or operation – Value normalized to [0.0, 1.0] ^{Note 1}

Note 1: Uses Eq 6.1: $f(x) = 2 * (1/(1 + k^x) - 0.5)$ with $k = 1.01$

Table VI-4. Discriminator Values Used for Syntactic Correlation (continued) (After [She02])

Number	Discriminator	Value to Vector
Frequency of Occurrence		
17	minOccurs	If Optional – 0.0 Otherwise – 1.0
18	maxOccurs	If not specified – Raw value = 1, normalized to [0.0, 1.0] ^{Note 1} Otherwise - Value normalized to [0.0, 1.0] ^{Note 1}
Data Size Specification		
19	minLength	If operation, attribute not string, or minLength not specified – 0.0; Otherwise - Value normalized to [0.0, 1.0] ^{Note 1}
20	maxLength	If operation, attribute not string, or maxLength not specified – 0.0; Otherwise - Value normalized to [0.0, 1.0] ^{Note 2}
21	totalDigits	If operation, attribute not bigDecimal, or totalDigits not specified – 0.0; Otherwise - Value normalized to [0.0, 1.0] ^{Note 2}
22	fractionDigits	If operation, attribute not bigDecimal, or fractionDigits not specified – 0.0; Otherwise - Value normalized to [0.0, 1.0] ^{Note 2}
Data Value Constraints		
23	pattern	If Atomic – If pattern defined – 1.0 Else 0.0 For Complex Attribute – Value normalized to [0.0, 1.0] ^{Note 1}
24	numEnumerations	If operation or numEnumerations not specified – 0.0; Otherwise, Value normalized to [0.0, 1.0] ^{Note 1}
25	minExclusive	If operation, attribute not numeric, or minExclusive not specified – 0.0; Otherwise - Value normalized to [0.0, 1.0] ^{Note 2}
26	maxExclusive	If operation, attribute not numeric, or maxExclusive not specified – 0.0; Otherwise - Value normalized to [0.0, 1.0] ^{Note 2}
27	minInclusive	If operation, attribute not numeric, or minInclusive not specified – 0.0; Otherwise - Value normalized to [0.0, 1.0] ^{Note 2}
28	maxInclusive	If operation, attribute not numeric, or maxInclusive not specified – 0.0; Otherwise - Value normalized to [0.0, 1.0] ^{Note 2}

Note 1: Uses Eq 6.1: $f(x) = 2 * (1/(1 + k^{-x}) - 0.5)$ with $k = 1.01$

Note 2: Uses Eq 6.3: $f(x) = \log(x + 1)/5$

As shown in Figure VI-1, the back-propagating neural net used in the OOMI IDE consists of a totally connected network of nodes that can be divided into three layers. The first layer, the input layer, contains N nodes, corresponding to the N discriminators chosen to characterize the attributes and operation parameters from a *CCR* or *FCR Schema*. The third layer, the output layer, contains M nodes, corresponding to the M attribute and operation parameters contained in the *FCR Schema*. The second, middle, layer is a hidden layer consisting of $(N + M) / 2$ nodes connecting the input and output layers used in training the network.

The network is trained by first assigning nominal weights to the edges of the network, which is done automatically by the training algorithm. Then, the discriminator vectors constructed for the attribute and operations of the neural net's corresponding *FCR Schema* are input to the network and an output determined for the specified edge weights. This output is compared with the desired output for each attribute or operation. The desired output for an attribute or operation consists of a unique vector of M zeroes and ones, where M is the number of attributes and operations contained in the *FCR Schema*. The edge weights are adjusted and the training effort continued until the actual output is within tolerance of the desired result, as specified by the user-entered threshold value. Once the error is reduced below the threshold value, the network is considered trained and it is saved in the FCR Syntax's *Net* subcomponent for later use during component and federation class correlation.

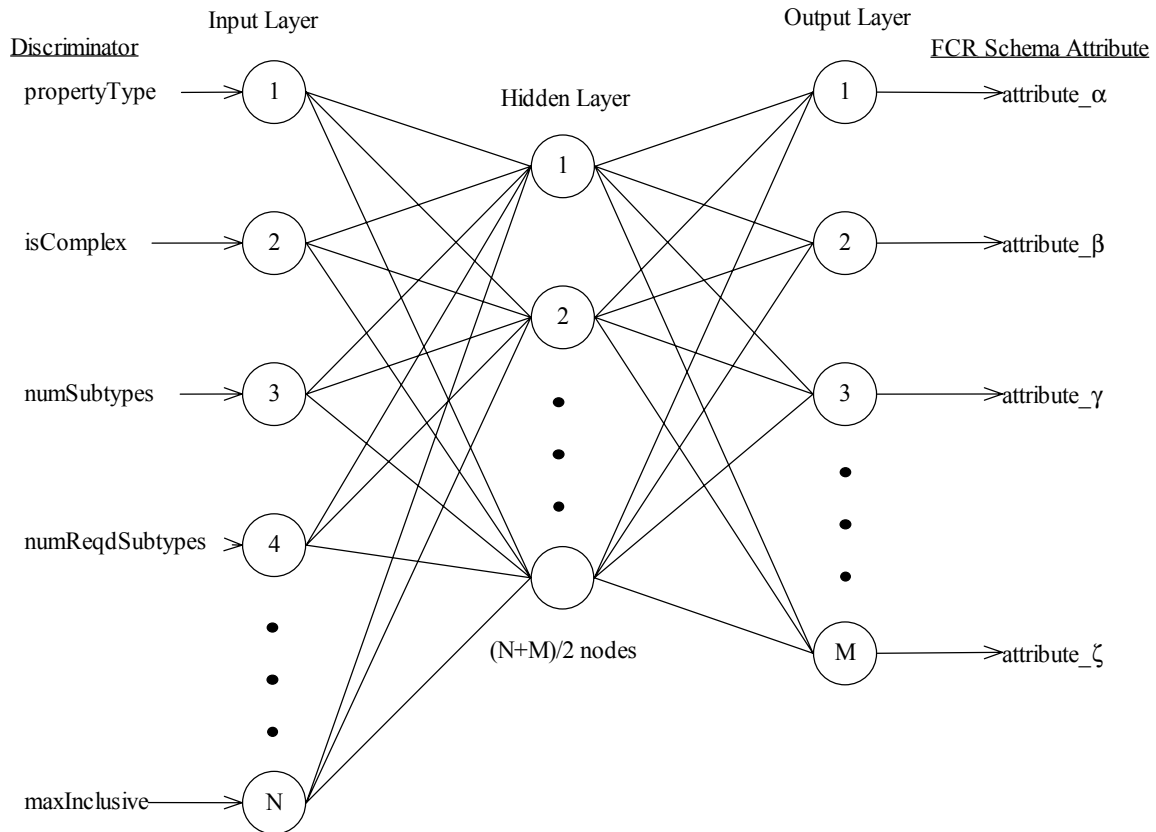


Figure VI-1. Back-Propagation Neural Network Architecture in OOMI IDE (After [LC00])

neural network is input with the discriminator vector for an attribute or operation parameter, the edge weights of the net will be adjusted until the output matches the target value, within the threshold value tolerance. This process will be repeated for the remaining attributes and operation parameters until the network has been trained for all inputs. At this point the network is considered trained and can be used to recognize other attributes or operation parameters having the same data content and structure.

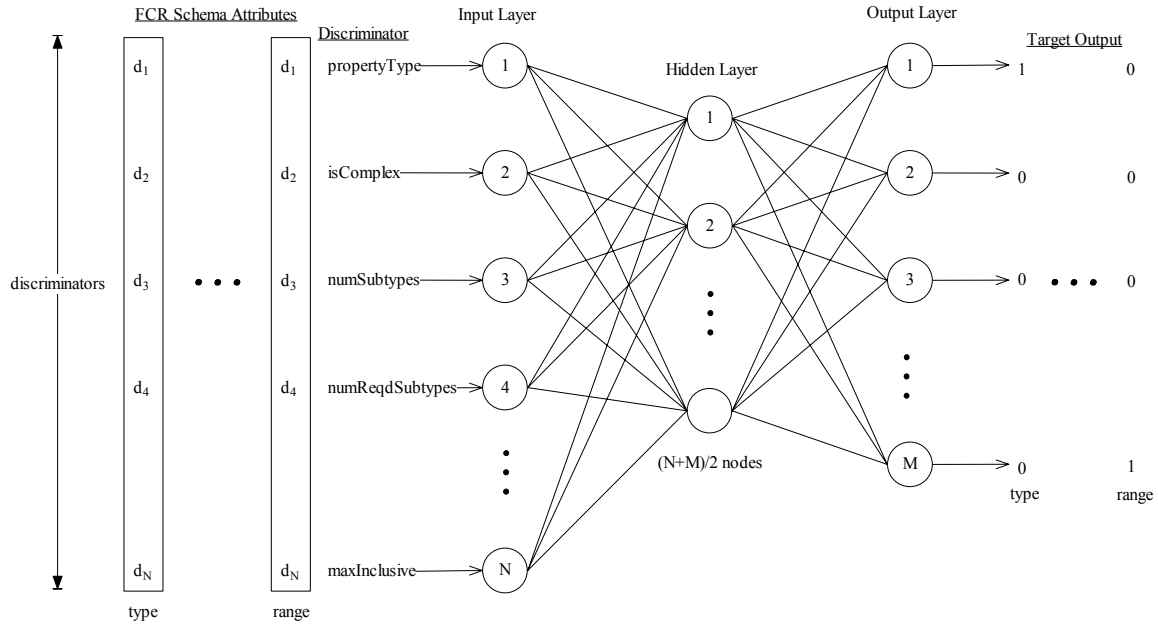


Figure VI-3. Training OOMI IDE Neural Networks (After [LC00])

2. Using Syntactic and Semantic Information to Correlate Component and Federation Representations of Real-World Entities

Using the syntactic and semantic information generated from the CCR and FCR Schemas, the OOMI IDE assists the interoperability engineer in establishing a correspondence between a component representation and the federation representation of the real-world entities involved in system interoperation. Access to the correlation functionality is provided via the OOMI IDE *Correlation Window* during the *Register CCR* phase of FIOM development. The *Correlation Window* outlines a two-step process for locating federation entities whose FCR Schema corresponds to the CCR Schema for a component system class being registered.

a. Semantic Correlation Process

The first step of the correlation process, semantic correlation, screens FEVs using semantic information contained in their corresponding FCRs. The user initiates the semantic correlation process by setting the semantic correlation threshold to the desired value and then selecting the CCR for which a match is desired. The user can adjust the semantic correlation threshold value to selectively set the level for displaying potential matches. Once the threshold value is set and the user selects “*Filter Using Keywords*”, the OOMI IDE will retrieve the *CCR Semantics* component for the CCR selected and sequentially examine the *FCR Semantics* component for each FEV in the FIOM. From the *FCR Semantics* component the semantic correlator will obtain the count of the number of keywords in the *CCR Semantics* component matching a keyword in the *FCR Semantics* component. The IDE will then normalize the count as the ratio of *CCR Semantics* keywords matching FEV FCR Semantics keywords to the total number of CCR Semantics keywords. This count will then be saved as the FEV keyword score and the process repeated for the next FEV. After all the FEVs have been examined, the FEVs will be ordered according to keyword match score, and the ordered list of FEVs whose keyword match score exceeds the threshold value will be output to the IDE *Correlation Window*.

b. Syntactic Correlation Process

Following the semantic correlation process, the syntactic correlation algorithm attempts to find an FEV corresponding to the selected CCR using the trained neural network associated with each FEV. The user will initiate the syntactic correlation process by setting the syntactic correlation threshold to the value desired and then selecting the FEVs returned by the previous keyword match process to be considered by the syntactic matching routine. The user can adjust the syntactic correlation threshold value to selectively set the level for displaying potential matches. Once the threshold value is set and the user selects “*Filter Using Neural Net*”, the OOMI IDE will retrieve the *CCR Syntax Vector* subcomponent for the CCR selected by the user.

For each FEV selected for review, the OOMI IDE will use the discriminator vectors contained in the *CCR Syntax Vector* subcomponent as input to the neural network previously saved in the FEV’s *FCR Syntax Net* subcomponent. The

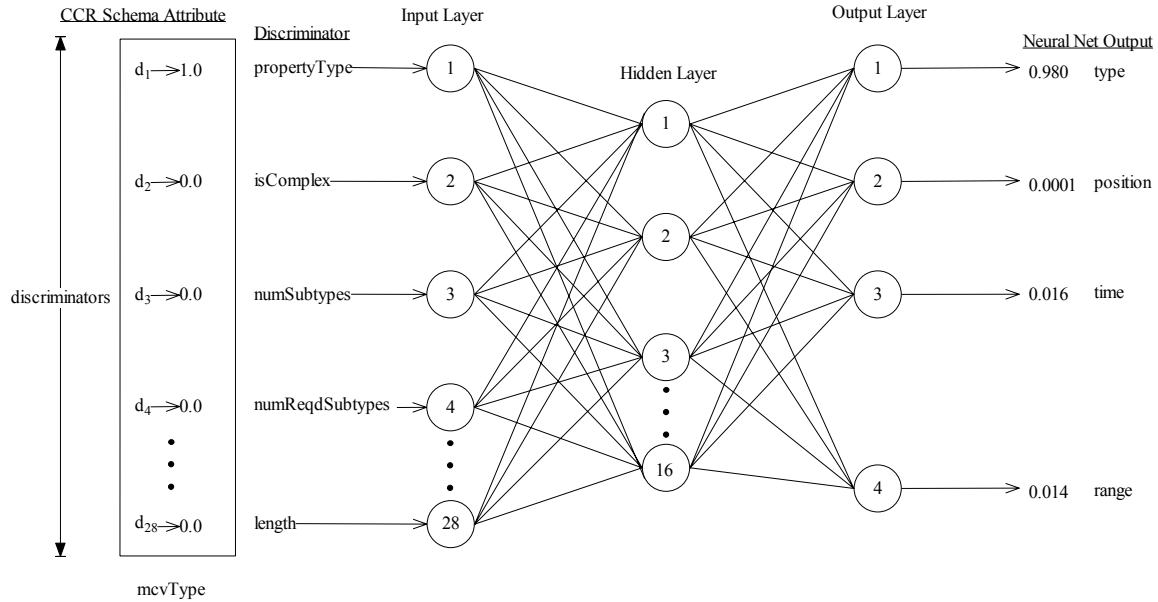


Figure VI-5. Using Trained Neural Network to Evaluate Attribute and Operation Correspondence (After [LC00])

The process is repeated, comparing each CCR Schema attribute and operation with every FCR Schema attribute and operation, resulting in a *CCR-FCR Comparison Matrix* containing scores of the comparison. Figure VI-6 illustrates the CCR-FCR Comparison Matrix comparing the MechanizedCombatVehicle CCR to the GroundCombatVehicle_View1 FCR.

		FCR Schema Attributes			
		type	position	time	range
CCR Schema Attributes	mcvType	.980	.0001	.016	.014
	mcvLocation	.0007	.980	.020	.096
	mcvTime	.003	.972	.027	.013
	mcvRadius	.012	.014	.0003	.985

Figure VI-6. Example CCR-FCR Comparison Matrix

This process is repeated for the remaining FEVs selected for comparison with the CCR. In order to facilitate comparison among FEVs as to which might offer the closest match to the CCR being registered, a single value is computed for the CCR-FCR Comparison Matrix. Determining the maximum value from each row of the CCR-FCR Comparison Matrix and then computing the 2-norm of the resultant maximums provides this single value. The ratio of this result to the 2-norm of the row maximums for a perfect CCR-FCR match is then saved for comparison with other FCRs. The 2-norm or length of a vector is computed by taking the square root of the sum of the squares of each element in the vector [Ant94].

By using the row maximum, the FCR with the highest score (i.e. closest match) for a CCR attribute or operation parameter, will contribute a higher value toward the overall FCR score, regardless of which FCR attribute or operation parameter provided the closer match. The 2-norm preserves the relative score between two FCR's regardless of which attribute or operation parameters contributed to the score, i.e., an equal score is provided to two FCRs whose maximum attribute or operation parameter scores are equal. In addition, the 2-norm provides a higher score to an FCR which provides a perfect or near-perfect match for one or more CCR attributes or operation parameters than to an FCR which doesn't provide a close match with any of the CCR attributes or operation parameters, even though the sum of the maximum attribute or operation parameter scores may be equal. Figure VI-7 illustrates computation of the score for the CCR-FCR Comparison Matrix using the process outlined above.

		FCR Schema Attributes				Row Maximum (X)	$\sum X^2$
CCR Schema Attributes	mcvType	0.980	0.0001	0.016	0.014	0.980	0.961
	mcvLocation	0.0007	0.980	0.020	0.096	0.980	0.960
	mcvTime	0.003	0.972	0.027	0.013	0.972	0.945
	mcvRadius	0.012	0.014	0.0003	0.985	0.985	<u>0.971</u>
							$\sqrt{\sum X^2}$
							1.959
							(as percent of maximum 2-norm) 97.93

Figure VI-7. Computing Single Value for CCR-FCR Comparison Matrix

After all FEVs have been examined, the FEVs will be ordered according to CCR-FCR Comparison Matrix score, and the ordered list of FEVs whose score exceeds the threshold value will be output to the IDE *Correlation Window*. In addition, the average of the semantic and syntactic scores is displayed in the Correlation Window to further indicate similarity of the CCR being registered and the FCRs chosen for review.

Results from the syntactic and semantic correlation methods are used as an aid to the interoperability engineer for determining when component and federation representations refer to the same real-world entity. Final determination of whether component and federation representations correspond is the responsibility of the interoperability engineer.

C. SUMMARY

The correlation methodology implemented for the OOMI IDE is used to assist the interoperability engineer in adding a Component Class Representation (CCR) to a Federation Interoperability Object Model (FIOM) during the *Register CCR* phase of IDE operation. Assistance is provided to the interoperability engineer in terms of computer aid for finding the Federation Entity (FE) corresponding to the same real-world entity modeled by the CCR. The OOMI IDE correlation methodology uses a two-phased approach for establishing this correspondence. In the first phase, semantic information taken from keywords used to describe component and federation models of a real-world entity is used to establish the correspondence. In the second phase, details about the structure and composition of the attributes and operations used to model the real-world entity are used to correlate component and federation models. Potential correspondences between a CCR and an FCR are provided in terms of a score for both the semantic and syntactic phases of the correlation effort, or a combination of the two. Comparison of scores among potential FCR matches will direct the interoperability engineer toward the most likely match for a CCR. However, final determination of CCR-FCR correspondence requires a one-to-one correspondence between the attribute and operation sets of a potential CCR-FCR match as discussed in Section V.C.3.b. Determination of attribute and operation correspondence and operation behavioral equivalence is the responsibility of the interoperability engineer and is not automated in the OOMI IDE.

Implementation of the correlation methodology detailed in this chapter was initiated by Pugh [Pug01] and continued under Shedd [She02]. While implementation of the correlation methodology has not been completed, thereby precluding an assessment of the method's effectiveness, criteria for conducting such an assessment can be provided. The primary criteria to be used for such an assessment are a determination of the precision and recall attained by the search for the federation model of a real-world entity corresponding to a component model being registered. In Section III.A precision is defined as the ratio of the number of objects correctly correlated and the total number of objects correlated, and recall is defined as the ratio between the number of objects correctly correlated and the number of correct correlations possible. As applied to the real-world entity model correlation problem, a calculation of precision will provide the ratio of the number of FCR's returned by the correlation methodology that are correct matches for the CCR being registered to the total number of FCR's returned as candidate matches. Recall will provide a ratio of the number of FCR's returned by the correlation methodology as correct matches for the CCR being registered to the number of FCR's that are correct matches for the CCR being registered. As discussed in Sections VI.B.2.a and VI.B.2.b the interoperability engineer can set a threshold for display of candidate matches returned by either the semantic or syntactic correlation methodologies or by a combination of the two methods. Values for precision and recall should be provided with a range of threshold values used for returning candidate matches for each of these possible alternatives.

VII. OBJECT-ORIENTED METHOD FOR INTEROPERABILITY (OOMI) TRANSLATOR

A. TRANSLATOR OVERVIEW

Interoperability was previously defined as the ability to exchange information and tasks between systems [LISI98, Pit97]. As mentioned in Section II.A.1, differences in perspective, modeling constructs used, and application design specifications result in heterogeneous modeling of the information and tasks to be shared among systems. Resolution of these heterogeneities is required in order to enable system interoperation.

The purpose of the translator presented in this chapter is to resolve heterogeneities among systems in order to enable their interoperation. As discussed in Section IV.C.1.a, these heterogeneities can be categorized as either differences in *view*, indicating that two systems have a different perspective on the characteristics required to model a real-world entity, or differences in *representation* when systems differ on how those characteristics are modeled. Under the Object-Oriented Method for Interoperability (OOMI), an interoperability engineer captures differences in view and representation of the real-world entities involved in the interoperation among systems in a Federation Interoperability Object Model (FIOM). An FIOM is created for a specified federation of systems prior to runtime using a specialized toolset, the OOMI Integrated Development Environment (IDE) previously described in Chapter V.

The real-world entities involved in the interoperation among systems are modeled as Federation Entities (FEs) in the OOMI. Differences among federation components on what should be included in the model of these real-world entities are captured in the form of one or more views for each FE, termed Federation Entity Views (FEVs) in the OOMI. Differences in representation of the attributes and operations that comprise a view among federation components are captured as various Component Class Representations (CCRs) of the FEV. Each FEV also includes a standard representation of the view that is termed a Federation Class Representation (FCR). FEVs are related in terms of their common attributes and operations by means of an FCR Schema Inheritance Hierarchy that is defined for each FE.

The OOMI *translator* uses information contained in the FIOM created prior to runtime for resolving heterogeneities among federation systems at runtime. The translator uses the FCR Schema Inheritance Hierarchy to resolve differences in view among systems, as caused by heterogeneities of scope, level of abstraction, and temporal validity. The translator uses an *FCR-CCR Translation* class associated with each component representation of a view (CCR) to resolve heterogeneities of hardware and operating systems, organizational models, structure, presentation, and meaning. Figure VII-1 illustrates the interaction between the translator and FIOM for a wrapper-based implementation of the translator on both source and destination systems.

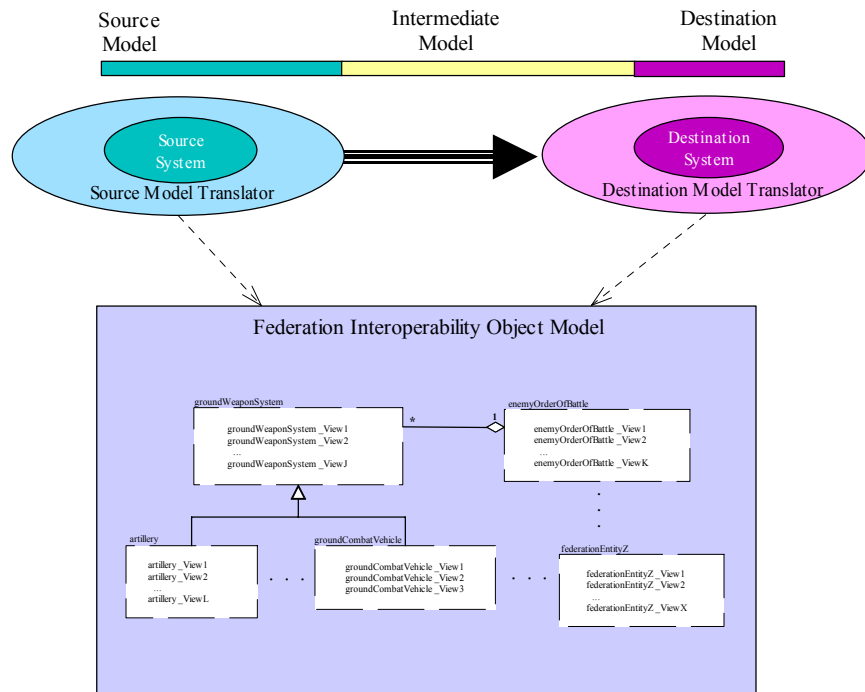


Figure VII-1. Source and Destination System Translator Implementation

B. TRANSLATOR ARCHITECTURAL ALTERNATIVES

Translator implementation can be adapted to accommodate a number of federation architecture alternatives. In one implementation, the translator can be employed as part of a software wrapper that logically envelops a component system. The wrapper would function to intercept incoming and outgoing information from the wrapped system and convert it from one model to another. In another implementation the

translator could be realized as part of middleware that resides on a separate platform between the source and destination systems. Translator placement determination is not a topic of this dissertation; however, this section provides substantiation of translator architectural compatibility.

In a wrapper-based implementation, translator functionality could be included in a wrapper around either the source or destination systems, or around both source and destination systems. Implementing the translator functionality as a wrapper around only the source system would require the wrapper to convert the outgoing information from the source model to the destination model prior to forwarding to the destination system. A wrapper would not be required for the destination system as information received at that system would already be provided in the destination model.

A source-system-only wrapper is implemented by incorporating both the source and destination model translators in the source system wrapper as depicted in Figure VII-2. In this illustration the source model translator intercepts outgoing messages from

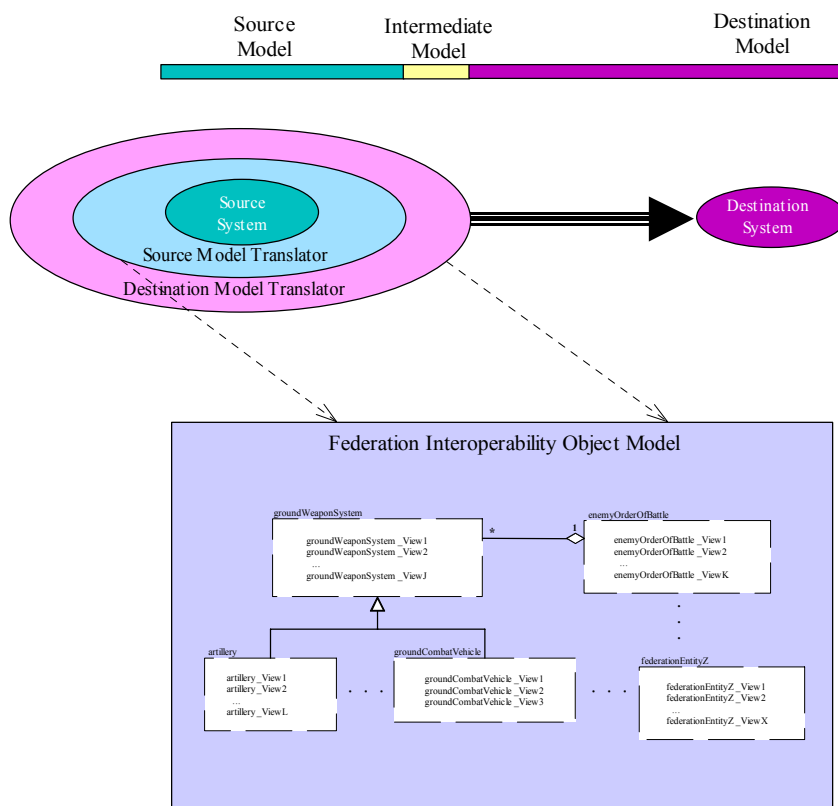


Figure VII-2. Source-System-Only Translator Implementation

the source system and converts them from the source system model to an intermediate model. This output is then forwarded to the destination model translator, also contained in a wrapper surrounding the source system, where it is converted from the intermediate model to the destination system model prior to forwarding to the destination system. A source-system-only implementation could be accomplished without the use of an intermediate model, converting directly from the source to destination model; however, for a federation of n systems the number of required translations would increase from the $2n$ required with the use of an intermediate model to $n(n-1)$ required without.

Similarly, implementing the translator functionality as a wrapper around only the destination system would require the destination wrapper to convert incoming information from the source to the destination model, possibly involving translation to an intermediate model in the process. In this implementation, a wrapper would not be required for the source system as the destination wrapper performs all data transformation. Figure VII-3 illustrates a destination-system-only wrapper implementation.

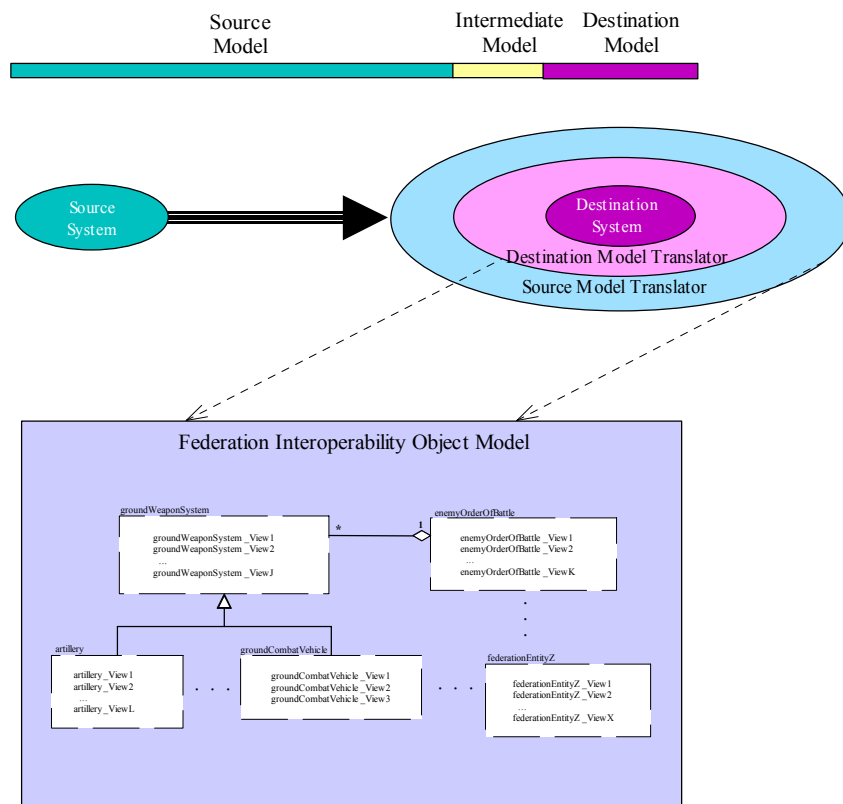


Figure VII-3. Destination-System-Only Translator Implementation

An additional architectural alternative would be to provide source model translation functionality in a wrapper surrounding the source system and destination model translation functionality in a wrapper surrounding the destination system. The source wrapper translation functionality would convert outgoing information from the source model to an intermediate model for transmission to the destination system. The destination wrapper translation functionality would convert the incoming information from the intermediate model used for transmission to the model expected by the wrapped system. This architecture alternative is shown in Figure VII-1.

Another alternative for translator implementation is to provide both the source and destination model translation functionality as part of middleware that resides on a separate hardware platform between the source and destination systems in a hub-and-spoke architecture implementation. All information exchanged among federation systems would be routed from the source system to the hub where resident middleware would convert the information from the source to the destination model prior to forwarding it to the appropriate destination. Figure VII-4 depicts this architectural alternative.

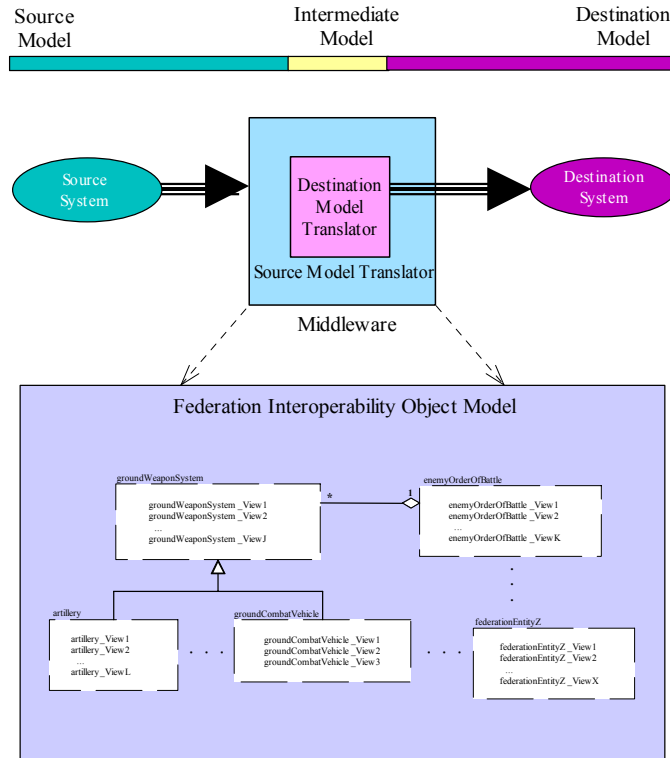


Figure VII-4. Middleware Translator Implementation

C. TRANSLATOR FUNCTION

The translator uses the FIOM created prior to runtime to dynamically resolve differences in information shared among systems. As discussed in Section IV.C.1.a, modeling differences for shared information consist of either differences in view or differences in representation of a view for the real-world entity modeled. Differences in *view* are resolved through exploitation of the information contained in the FCR Schema Inheritance Hierarchy for the federation model of the real-world entity using Liskov and Wing's notion of behavioral subtyping [LW94]. Differences in *view representation* are resolved by use of the translations included with the FCR-CCR Translation class associated with each component system model of the real-world entity. As indicated in Section IV.C.1.b(2), use of an intermediate representation for a real-world entity reduces the number of translations required for a system with n representations of that entity from $n(n-1)$ to $2n$. Therefore, the OOMI translator uses a two-step process, involving the use of an intermediate representation, to reduce the number of required translations. In the first step the source model of a real-world entity is translated to an equivalent intermediate model of that entity. In the second step, this intermediate model is translated to a behaviorally equivalent model suitable for use by the destination system.

In order to illustrate the translator functionality described in this chapter, a continuing example is used to demonstrate translator action when exporting an instance of a real-world entity from one system to another. For the example we use the *ground combat vehicle* real-world entity introduced in Figure IV-2 and the different models used to portray that entity on a hypothetical federation of systems. We describe how an instance of the real-world entity exported from one system is transformed for use by an application on another system in the federation that presents a different model of the real-world entity than that provided by the source system. For our example System B will be the source and System D the destination.

1. Source To Intermediate Model Translation

A source model of a real-world entity is captured in the form of a CCR in the FIOM. As discussed in Section IV.C.1.b(2), when defining a Federation Entity to depict the information and operations shared between systems in a federation, for each component representation of a real-world entity (CCR) there will be one, and only one,

FCR defined for that real-world entity. In addition, the FCR will be defined such that there is a one-to-one correspondence between the CCR and FCR Schema attribute and operation sets. Thus a CCR and its corresponding FCR share the same view of a real-world entity. Therefore, translation from source to intermediate model of a real-world entity requires only resolution of any differences in representation between the models.

The OOMI defines an object model of the real-world entities involved in the interoperation among systems, the FIOM. The FIOM is constructed for a system federation by extracting information contained in the component systems' external interfaces. As indicated in Section V.D.2.a(2), specification of a component system's external interface is provided in terms of an XML Schema description for each of the real-world entities whose information or operations are shared by the system. The OOMI IDE uses these XML Schemas in constructing the FIOM for a system federation. The export and import of information required to accomplish information exchange and joint task execution is achieved through the use of XML instance documents conforming to these schemas. Communications among systems required to accomplish information exchange and joint task execution are then achieved through the export and import of messages in the form of XML instance documents conforming to these schemas.

Source to intermediate model translation first involves conversion of the information exported by a source system from the source model format to an object representation of that information for use by the OOMI translator. This step involves converting the information from the XML instance document representation of the exported message to a corresponding object representation of that message. The next step involves conversion from the source object representation to the corresponding intermediate object representation. The final optional step entails converting the intermediate object representation to an intermediate XML instance document representation if the federation architecture uses XML for transferring information among systems.

a. Converting From XML to Object Representation of Exported Information

Converting an exported message from an XML instance document representation to a corresponding object representation is done using XML data binding [BOD01]. For each message type exported from or imported to a component system, an

XML Schema is used to define the allowable message contents. The OOMI IDE generates equivalent class representations of these XML Schemas using data binding, which it stores in the FIOM as *CCR Schemas*. As part of a generated class, the data binding process automatically creates an *unmarshal* method used to convert an XML instance document, conforming to the XML Schema used to generate the class, to an instance of the generated class. Figure VII-5 illustrates the process for converting a source XML instance document to its equivalent CCR Schema object and the relationship between the XML instance document's governing XML Schema and the CCR Schema object's defining CCR Schema.

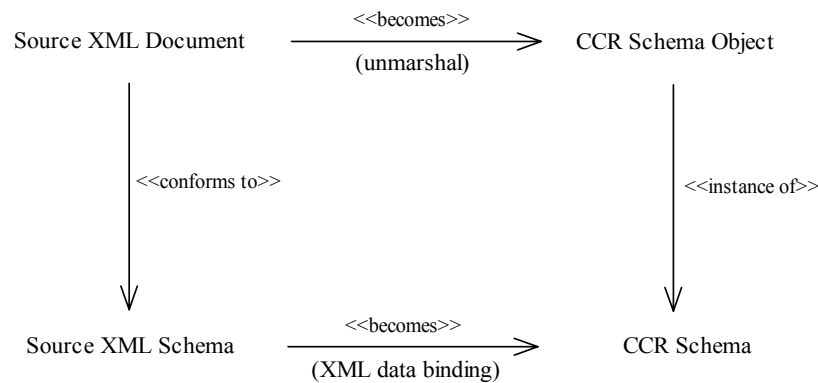


Figure VII-5. Process for Converting Source XML Instance Document to its Equivalent CCR Schema Object

In order to convert an exported XML instance document to its equivalent object representation, the translator must first determine which unmarshal method to use for the conversion. The correct unmarshal method to use is the one contained in the CCR Schema generated from the XML Schema to which the exported XML instance document conforms. Location of the CCR Schema that corresponds to the exported source XML instance document can be accomplished using one of three methods: 1) use of a standard convention for naming XML instance documents, XML Schemas, and generated CCR Schemas; 2) examination of the SchemaLocation attribute of the instance document root element to determine the XML Schema to which the instance document conforms, using a naming convention for determining the CCR Schema given the XML Schema; or 3) use of a common, unique XML namespace Uniform Resource Identifier (URI) for both the

XML Schema and instance document and saving that namespace URI with the CCR that contains the generated CCR Schema for future reference.

The third method above is the one chosen for use in the initial prototype OOMI IDE and translator. By saving the XML namespace URI of the XML Schema used to generate a CCR Schema with its containing CCR, the *xmlns* attribute of a received XML instance document can be used to locate the appropriate CCR Schema. For example, Figure VII-6 depicts an XML document for an instance of the System B MechanizedCombatVehicle shown in Figure IV-3. As seen in the figure, the namespace URI (targetNamespace) for the XML instance document is "http://nps.navy.mil/cs/oomi/-systemB/mechanizedCombatVehicle." Similarly, for the governing XML Schema (specified in the instance document's xsi:schemaLocation attribute) the namespace URI is also "http://nps.navy.mil/cs/oomi/systemB/mechanizedCombatVehicle" as shown in Figure VII-7. By including this namespace URI with the CCR Schema generated from the Figure VII-7 XML Schema excerpt, the appropriate unmarshal method can be found for converting the Figure VII-6 XML instance document to its equivalent object representation.

```
<?xml version="1.0" encoding="UTF-8"?>
<mechanizedCombatVehicle xmlns="http://nps.navy.mil/cs/oomi/systemB/mechanizedCombatVehicle"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://nps.navy.mil/cs/oomi/systemB
C:\Translator\OOMI\projects\testapps\mechanizedCombatVehicle.xsd">
  <mcvType>tank</mcvType>
  <mcvLocation>
    <utmZone>
      <eastWest>1</eastWest>
      <northSouth>A</northSouth>
    </utmZone>
    <mgrsEasting>B222</mgrsEasting>
    <mgrsNorthing>A111</mgrsNorthing>
  </mcvLocation>
  <mcvTime>
    <day>16</day>
    <hourTime>14</hourTime>
    <minuteTime>41</minuteTime>
    <localTimeZone>U</localTimeZone>
  </mcvTime>
  <mcvRadius>1200</mcvRadius>
</mechanizedCombatVehicle>
```

Figure VII-6. Source XML Document “mechanizedCombatVehicle.xml”

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://nps.navy.mil/cs/oomi/systemB/mechanizedCombatVehicle"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:systemB="http://nps.navy.mil/cs/oomi/systemB"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:element name="mechanizedCombatVehicle">
    <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation>A mechanizedCombatVehicle msgtype provides System B model of ground
          combat vehicle real-world entity for the example in Figure III-3.</xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element ref="systemB:mcvType"/>
        <xsd:element ref="systemB:mcvLocation"/>
        <xsd:element ref="systemB:mcvTime"/>
        <xsd:element name="mcvRadius" type="systemB:distanceInKmType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="mcvType">
    <xsd:annotation>
      <xsd:documentation/>
    </xsd:annotation>
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:minLength value="4"/>
        <xsd:maxLength value="16"/>
        <xsd:enumeration value="tank"/>
        <xsd:enumeration value="personnelCarrier"/>
        <xsd:enumeration value="reconVehicle"/>
        <xsd:enumeration value="unknown"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="mcvLocation">
    <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation/>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="utmZone" type="systemB:utmZoneType"/>
        <xsd:element name="mgrsEasting" type="systemB:mgrsEastingType"/>
        <xsd:element name="mgrsNorthing" type="systemB:mgrsNorthingType"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="mcvTime">
    <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation>The day of a month and timekeeping in hours and minutes of a calendar day,
          using the 24-hour clock system and an associated time zone.</xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="day" type="systemB:dayType"/>
        <xsd:element name="hourTime" type="systemB:hourTimeType"/>
        <xsd:element name="minuteTime" type="systemB:minuteTimeType"/>
        <xsd:element name="localTimeZone" type="systemB:localTimeZoneType"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="distanceInKmType">
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="1200"/>
    </xsd:restriction>
  </xsd:simpleType>
  .
  .
  .
</xsd:schema>

```

Figure VII-7. Source XML Schema “mechanizedCombatVehicle.xsd” Excerpt

Upon receipt of an XML instance document from the source system, the translator determines the corresponding CCR Schema and invokes its unmarshal method to convert the document to an equivalent object representation. The resulting CCR Schema object is provided as input to the next translator step.

To illustrate this process, System B exports an XML instance document describing an instance of its model of the GroundCombatVehicle real-world entity. The exported “mechanizedCombatVehicle.xml” XML instance document is captured by the source model translator and first converted to an equivalent object representation of the exported information. Figure VII-8 shows a UML representation of the object generated from the exported XML instance document depicted in Figure VII-6. In addition, the XML instance document’s governing XML Schema (shown in Figure VII-7) and the resultant object’s defining CCR Schema are included to complete the illustration.

b. Translation From Source Object Representation to Intermediate Object Representation

Following conversion of the exported information to an object representation, the information must be translated from the source model object representation, a CCR Schema instance, to an intermediate object representation, an *FCR Schema* instance. As described in Section IV.C.1.b(2), each CCR has exactly one FCR to which it corresponds. Likewise, each CCR Schema corresponds to exactly one FCR Schema. This FCR Schema is created during FIOM construction as described in Section V.D.2.a(1). From Figure IV-3, it is seen that the System B MechanizedCombatVehicle model of our ground combat vehicle real-world entity corresponds to view 1 of the real-world entity. Thus, the MechanizedCombatVehicle CCR Schema instance will be converted to its corresponding GroundCombatVehicle_View1 FCR Schema instance.

This translation is accomplished through the use of an FCR-CCR Translation class defined by the interoperability engineer during FIOM construction. For each component system representation of an FEV, the interoperability engineer defines an FCR-CCR Translation class containing *translate* methods used to convert between component and federation representations of the FEV, as described in Section IV.C.1.b(3). These methods are used to translate the object representation of a received XML instance document from a CCR Schema instance to an FCR Schema

instance. Figure VII-9 illustrates the process for translating a CCR Schema instance to an equivalent FCR Schema instance using the translate method from the FCR-CCR Translation class associated with the CCR Schema.

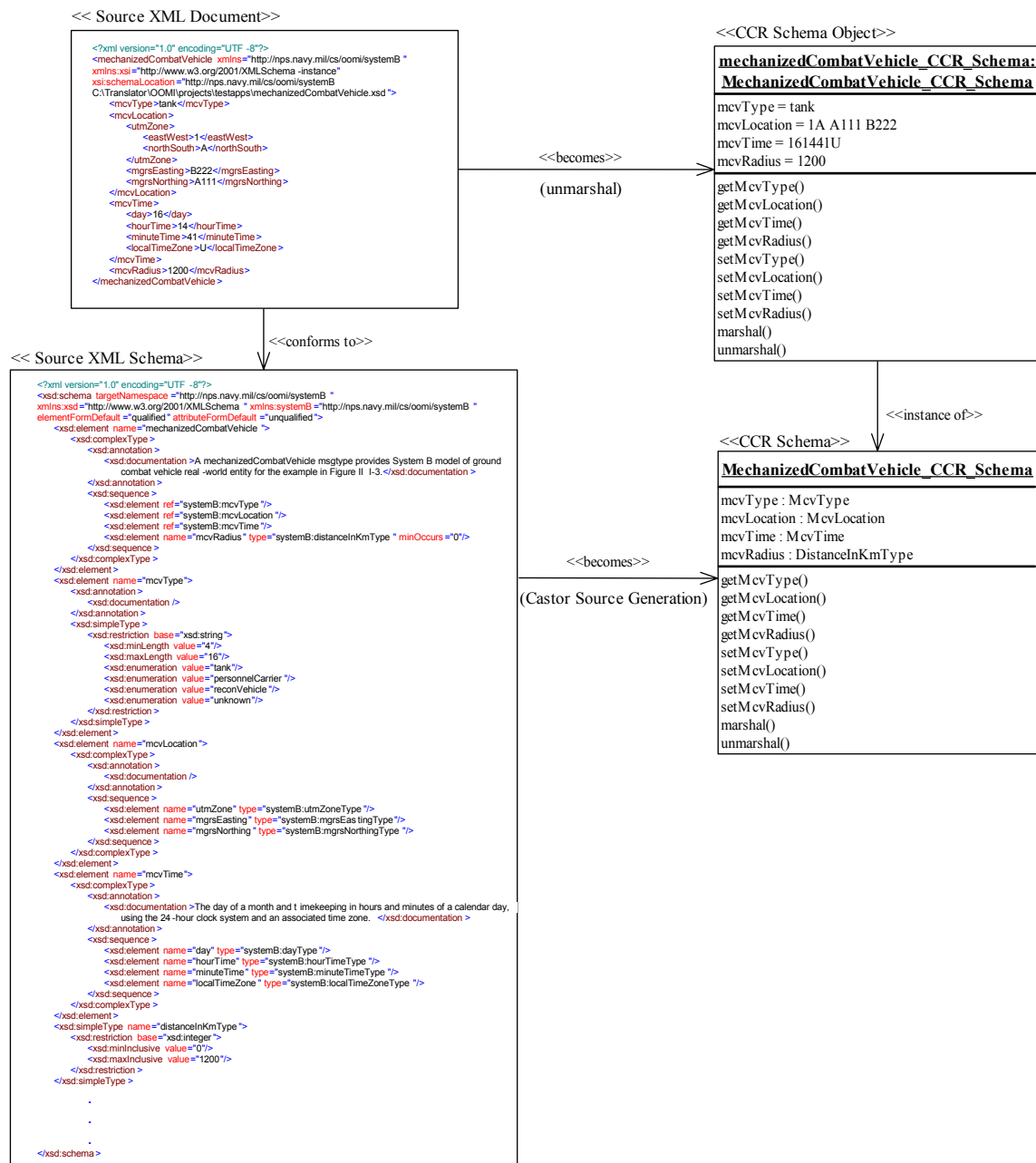


Figure VII-8. Converting From XML to Object Representation of Exported Information

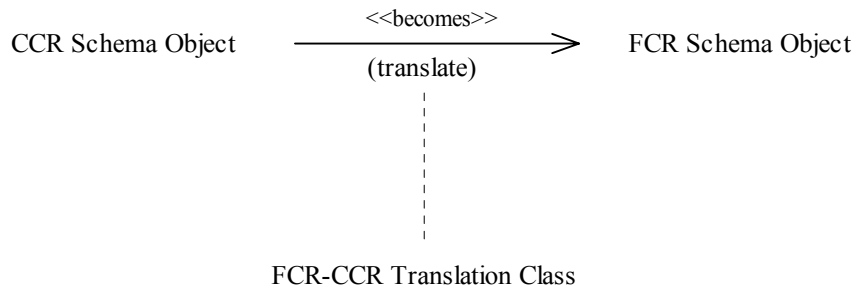


Figure VII-9. CCR Schema Object to FCR Schema Object Translation

For example, as previously mentioned, the `MechanizedCombatVehicle` class used to model our ground combat vehicle real-world entity on System B corresponds to view 1 of the real-world entity. Therefore, the `GroundCombatVehicle_View1__MechanizedCombatVehicle` Translation class would be used to convert from the source CCR Schema instance representation of our real-world entity to an equivalent intermediate FCR Schema instance representation. Figure VII-10 depicts how the `translate(mechanizedCombatVehicle : MechanizedCombatVehicle) : GroundCombatVehicle_View1` method from this class would be invoked to effect this translation. Then, depending on translator architectural implementation, this intermediate object representation could either be forwarded directly to the Intermediate-to-Destination-Model-translation or, optionally, converted into an XML Document representation of the intermediate object.

c. Converting From Intermediate Object Representation of Exported Information to XML Instance Document Representation

If the architecture used for translator implementation necessitates the use of an XML instance document for transporting information between source and destination systems, then it may be necessary to convert from the intermediate object representation of the exported information to an equivalent XML document representation. Such may be the case if the translator is implemented as a wrapper around both source and destination systems and an XML instance document is used for transporting information between systems as depicted in Figure VII-1.

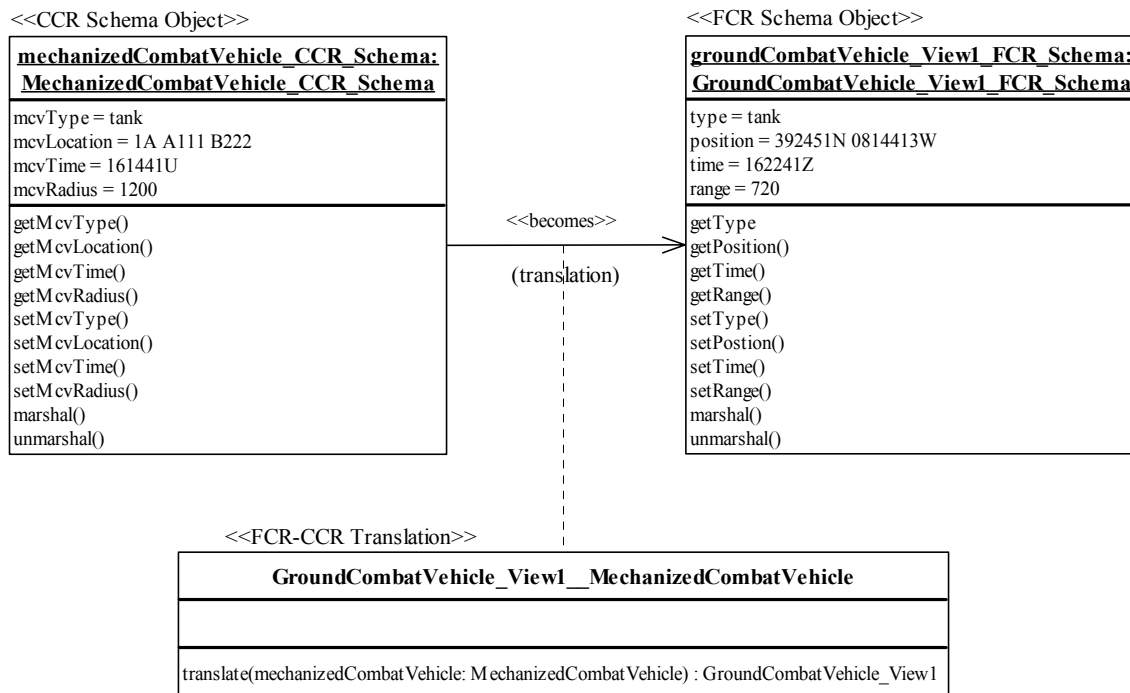


Figure VII-10. Converting from Source to Intermediate Object Representations

Conversion from an intermediate object representation to an XML instance document representation of the exported information is accomplished by use of the *marshal* method generated with the FCR Schema defining the intermediate object representation. XML data binding creates the marshal method for a class generated from an XML Schema to convert an instance of the class to its equivalent XML representation. If an intermediate XML document representation of the exported information were required, then the translator would use the intermediate FCR Schema object's marshal method to convert the object representation to its equivalent XML instance document representation. Figure VII-11 illustrates the process for converting an FCR Schema object to its equivalent intermediate XML instance document representation and the relationship between the FCR Schema object's defining FCR Schema and the XML instance document's governing XML Schema.

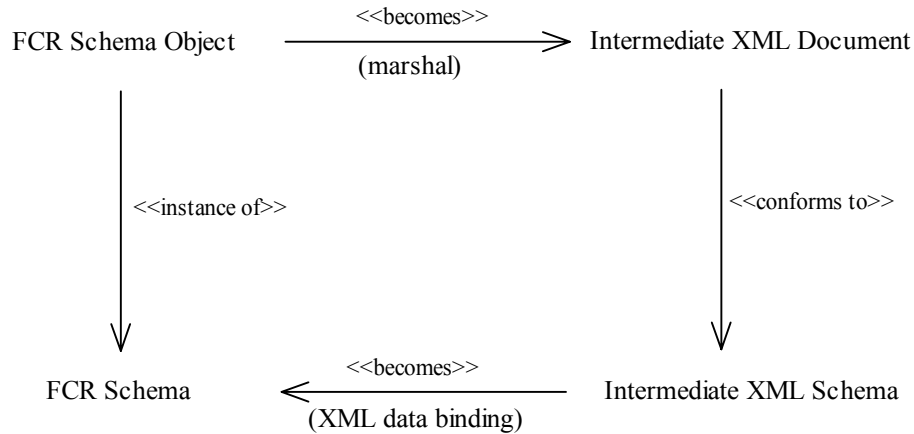


Figure VII-11. Process for Converting FCR Schema Object to its Equivalent XML Instance Document

As shown in Figure VII-12, the translator uses the marshal method defined for the GroundCombatVehicle_View1 FCR Schema from our continuing example to convert the groundCombatVehicle_View1 intermediate object representation of the exported information to its equivalent “groundCombatVehicle_View1.xml” XML instance document representation, detailed in Figure VII-13.

2. Intermediate To Destination Model Translation

Translation from an intermediate to a destination model of a real-world entity defining an interoperation requires resolution of potential differences in view as well as possible differences in representation between intermediate and destination models. Unless the source and destination systems have the same view of the real-world entity being modeled, the intermediate model created from the source model will have a different view of the modeled real-world entity than the destination model. Intermediate to destination model translation therefore first requires creation of an intermediate model with the same view as the destination model and then resolution of differences in representation between this new intermediate model and the destination model.

As discussed in Section IV.C.1.b(3), differences in view between intermediate and destination models of a real-world entity are resolved through use of the FCR Schema Inheritance Hierarchy defined for each FE in the FIOM. Differences in view representation between intermediate and destination models are resolved using the FCR-CCR Translation class associated with the destination system CCR.

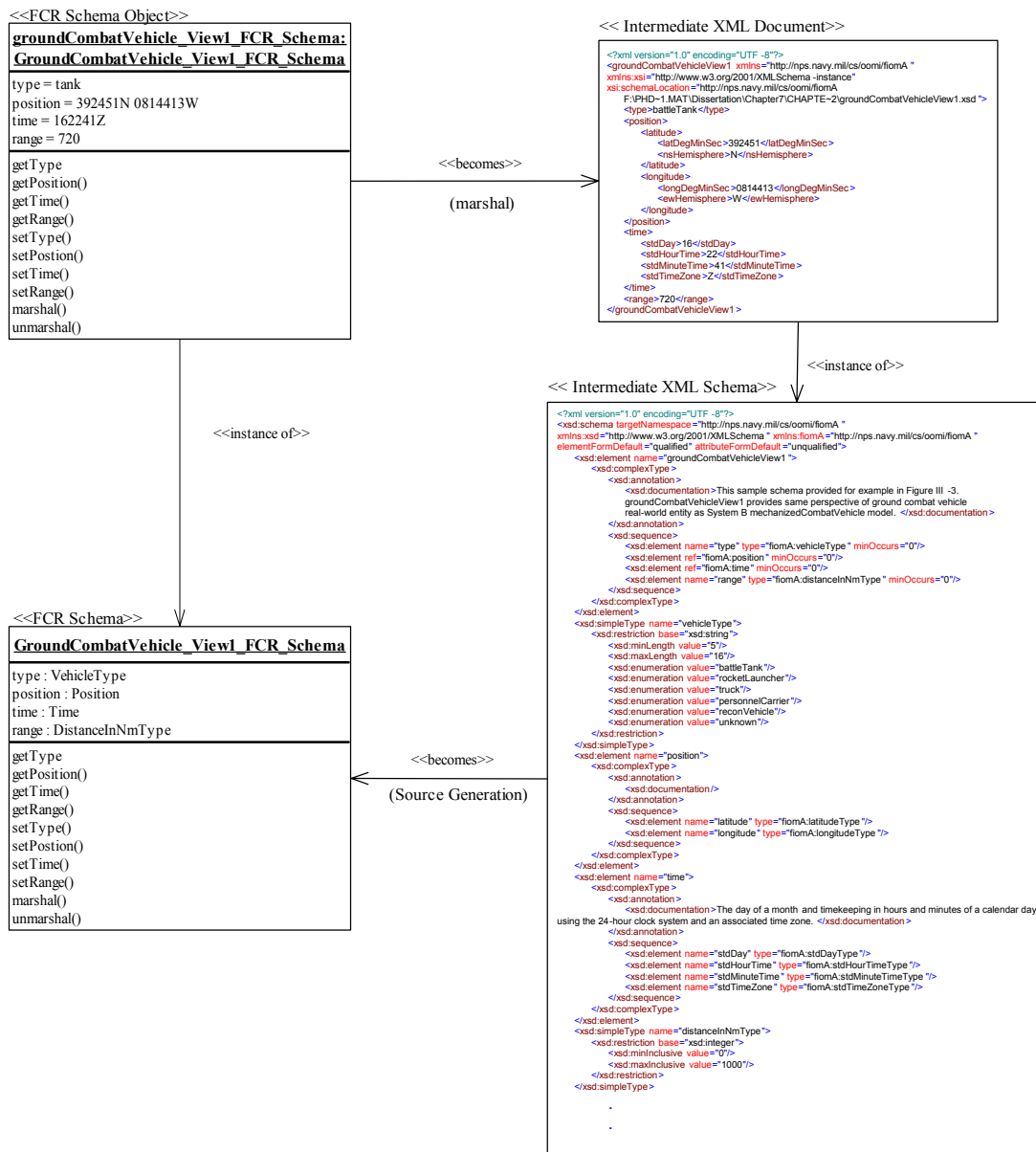


Figure VII-12. Conversion from Intermediate Object Representation to XML Document Representation

Converting from an intermediate to a destination model of a real-world entity involves four steps. First, if the translator receives the intermediate model in the form of an XML instance document, then it converts this document to its equivalent object representation. Second, the translator must resolve differences in view between the intermediate and destination models. Third, the translator must resolve differences in view representation between intermediate and destination models. Fourth, the destination

model must be converted from an object representation to an equivalent XML instance document representation for forwarding to the destination system application.

```
<?xml version="1.0" encoding="UTF-8"?>
<groundCombatVehicle_View1
xmlns="http://nps.navy.mil/cs/oomi/fiomA/groundCombatVehicle_View1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://nps.navy.mil/cs/oomi/fiomA
F:\PHD~1\MAT\Dissertation\Chapter7\CHAPTE~2\groundCombatVehicle_View1.xsd">
  <type>battleTank</type>
  <position>
    <latitude>
      <latDegMinSec>392451</latDegMinSec>
      <nsHemisphere>N</nsHemisphere>
    </latitude>
    <longitude>
      <longDegMinSec>0814413</longDegMinSec>
      <ewHemisphere>W</ewHemisphere>
    </longitude>
  </position>
  <time>
    <stdDay>16</stdDay>
    <stdHourTime>22</stdHourTime>
    <stdMinuteTime>41</stdMinuteTime>
    <stdTimeZone>Z</stdTimeZone>
  </time>
  <range>720</range>
</groundCombatVehicle_View1>
```

Figure VII-13. Intermediate XML Instance Document
“groundCombatVehicle_View1.xml”

a. Converting From XML Document Representation back to Intermediate Object Representation

Converting a received XML instance document representation back to its corresponding intermediate object representation reverses the process executed in Section VII.C.1.c. This would be necessary for a translator architecture where an XML instance document is used to transport information between a source and destination system. As seen in Figure VII-12, the received intermediate XML document is an instance of the XML Schema used to generate the FCR Schema defining the standard representation of the Federation Entity View. Converting the received XML instance document back to its equivalent object representation is done using an *unmarshal* method contained in the FCR Schema. The *unmarshal* method functions as the inverse of the *marshal* method previously discussed in Section VII.C.1.c. Both marshal and unmarshal methods are generated from an XML Schema during data binding. Figure VII-14

illustrates the process for converting the received intermediate XML instance document to its equivalent intermediate FCR Schema object and the relationship between the XML instance document's governing XML Schema and the FCR Schema object's defining FCR Schema.

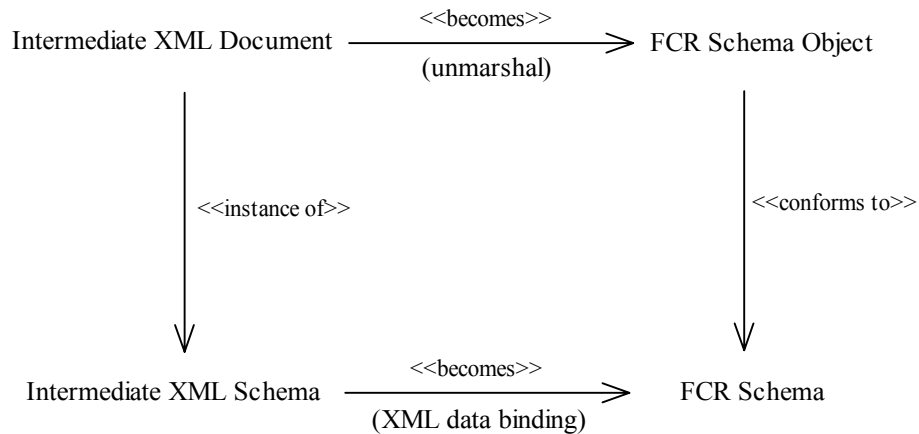


Figure VII-14. Process for Converting XML Instance Document to its Equivalent FCR Schema Object

The first thing the destination translator must do upon receipt of the intermediate XML instance document is to determine which unmarshal method to use to effect the conversion. This is accomplished in the same manner as was described in Section VII.C.1.a for converting an exported XML instance document to its corresponding object representation. As mentioned there, the OOMI IDE and translator use the XML namespace URI included with the intermediate XML instance document to locate the FCR and included FCR Schema instance containing the required unmarshal method. For the example “groundCombatVehicle_View1.xml” intermediate XML instance document seen in Figure VII-13, the “http://nps.navy.mil/cs/oomi/-FCR/groundCombatVehicle_View1” namespace URI is used to locate the GroundCombatVehicle_View1 FCR Schema containing the required unmarshal method.

Upon receipt of the XML instance document from the source model translator, the destination model translator determines the proper FCR Schema and invokes its unmarshal method to convert the document to its equivalent object representation. Figure VII-15 shows the groundCombatVehicle_View1 FCR Schema

object that results from applying the GroundCombatVehicle_View1 FCR Schema unmarshal method to the received “groundCombatVehicle_View1.xml” XML instance document representation.

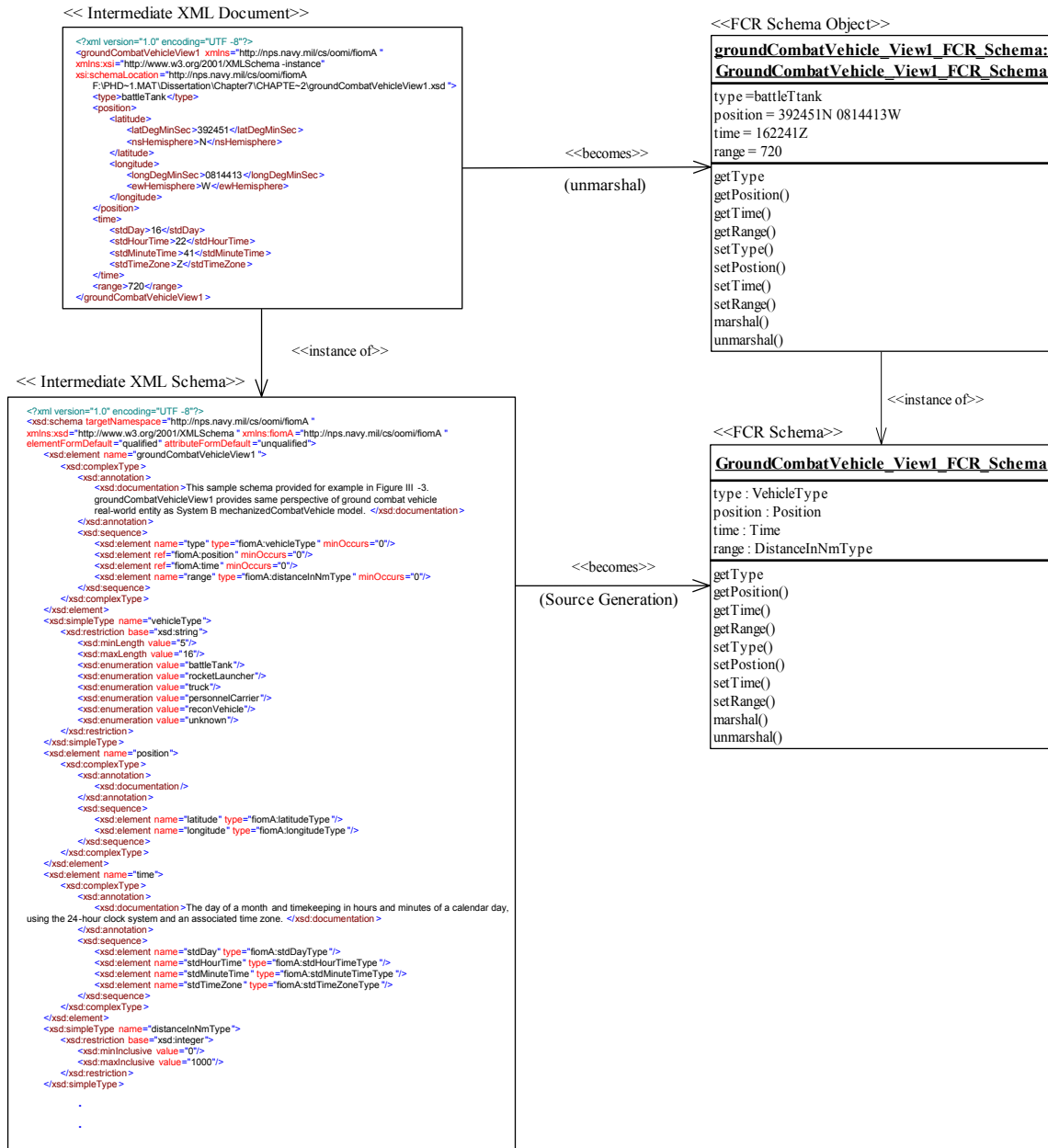


Figure VII-15. Conversion from XML Instance Document Representation back to Intermediate Object Representation

b. Resolving Differences in View between Received Intermediate Model and Destination Model of Real-World Entity

From the received intermediate FCR Schema object, the translator identifies the FCR Schema Inheritance Hierarchy containing the object's defining FCR Schema. This FCR Schema Inheritance Hierarchy is used to resolve differences in view between the received intermediate model and the destination model of the exported real-world entity. Differences in view are resolved by locating the destination system model of the real-world entity modeled by the received FCR Schema object and determining whether substitution of this received FCR Schema object would be behaviorally indistinguishable to destination applications expecting the destination system model of the entity. The substitution would be behaviorally indistinguishable to destination applications if all the mandatory attributes and operations expected for an object of the destination CCR Schema were contained in the received FCR Schema object and differences in representation of those properties could be resolved. The destination translator examines the FCR Schema Inheritance Hierarchy as well as the FCR Schema Inheritance Hierarchies of any supertype FEs to determine if such an FEV containing a CCR for the destination system exists. If so, the destination translator uses the information contained in the received FCR Schema object to create a new FCR Schema object whose attribute and operation sets correspond to those of the destination system's CCR Schema. The methodology for making that determination is described as follows.

The destination translator searches the FCR Schema Inheritance Hierarchy containing the received intermediate object's defining FCR Schema for an FEV containing a CCR for the destination system. If such a CCR is found, the destination CCR Schema is examined to determine if all mandatory properties (attributes and operations) contained in the CCR Schema have a corresponding property in the received intermediate object's defining FCR Schema. Determination of whether a property is mandatory is made by examining the *minOccurs* attribute of each property. This CCR Schema attribute is automatically set from the corresponding attribute contained in the XML Schema used for CCR Schema creation. A value for *minOccurs* of zero indicates the property is optional; otherwise it is regarded as mandatory.

If all of the mandatory properties contained in the destination CCR Schema do not have a corresponding property in the received FCR Schema object or if the FCR Schema Inheritance Hierarchy does not contain an FEV with CCR defined for the destination system, then the FIOM is searched to determine if a supertype of the FE containing the received FCR Schema object exists. If so, the translator searches the FCR Schema Inheritance Hierarchy of the supertype FE for an FEV containing a CCR for the destination system as previously described. If found, the destination system CCR Schema is examined to determine if all mandatory attributes and operations contained in the CCR Schema have a corresponding property in the received FCR Schema object. Then, if all mandatory properties contained in the CCR Schema have a corresponding property in the received FCR Schema object, a new FCR Schema object is created whose view corresponds to the destination system CCR Schema. If not, the process continues with the supertype of the FE currently being checked, until the FIOM root is reached.

If no FE is found with an FEV containing a CCR for the destination system such that all mandatory attributes and operations contained in the CCR Schema have a corresponding property in the received intermediate object's defining FCR Schema, then a translation failure is logged for interoperability engineer notification. The interoperability engineer may use logged translation failures to drive the requirements for future component system modification if the system federation requires the failed information or operation exchange.

From our continuing example, Figure VII-16 depicts an excerpt from the FCR Schema Inheritance Hierarchy for the ground combat vehicle real-world entity introduced in Figure IV-3. In our example, System B is exporting an XML document representation of an instance of its model of the ground combat vehicle real-world entity for import to System D, as captured in the file "mechanizedCombat-Vehicle.xml" seen in Figure VII-6. As seen in Sections VII.C.1.a and VII.C.1.b, the source System B translator has already converted the exported XML instance document to its equivalent object representation, and translated the component system object representation to an equivalent intermediate object representation. The next step in the process is for the destination model translator to translate this intermediate ground-

CombatVehicle_View1 object to a behaviorally equivalent intermediate model object representation. In order to accomplish this step the translator uses the FCR Schema Inheritance Hierarchy containing the received groundCombatVehicle_View1 object's defining FCR Schema.

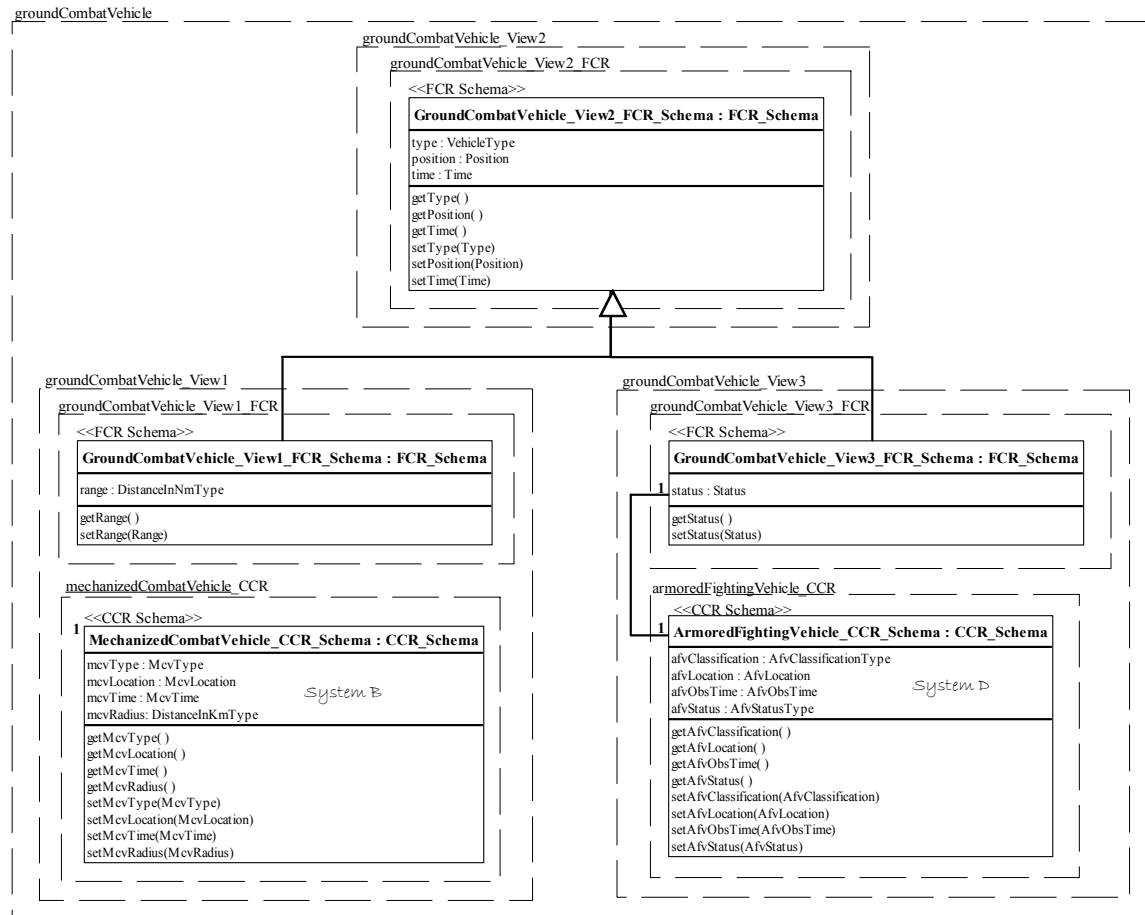


Figure VII-16. Example FCR Schema Inheritance Hierarchy Excerpt

From the Figure VII-16 FCR Schema Inheritance Hierarchy it can be seen that the GroundCombatVehicle_View3 FEV contains a CCR for the System D destination, ArmoredFightingVehicle CCR. Further investigation of the Armored-FightingVehicle CCR Schema shown in Figure VII-20 reveals that all of the mandatory attributes contained in the CCR Schema, *afvClassification*, *afvLocation*, and *afvObsTime*, have a corresponding property in the received object's defining GroundCombatVehicle_View1 FCR Schema, attributes *type*, *position*, and *time*, respectively. Therefore,

the destination translator uses the information contained in the received GroundCombatVehicle_View1 FCR Schema instance to create an instance of the GroundCombatVehicle_View3 FCR Schema corresponding to the destination ArmoredFightingVehicle CCR Schema. Associations between the CCR Schema and its corresponding FCR Schema attributes and operations established during FCR-CCR Translation class creation discussed in Section V.D.2.c are used in determining whether a CCR Schema's mandatory properties are satisfied by a received FCR Schema object.

c. Translation From Intermediate Object Representation to Destination Object Representation

Translation from the intermediate object representation to the destination model object representation is accomplished through the use of the translate method defined for the FCR-CCR Translation class associated with the destination CCR Schema determined by the translator as discussed in Section VII.C.2.b. The translate method defined for the selected FCR-CCR Translation class is applied to the FCR Schema object corresponding to the destination CCR Schema created from the received FCR Schema object as discussed in Section VII.C.2.b. Figure VII-17 depicts application of the FCR-CCR Translation class *translate* method to this new class, converting it from an intermediate object representation to the destination model object representation.

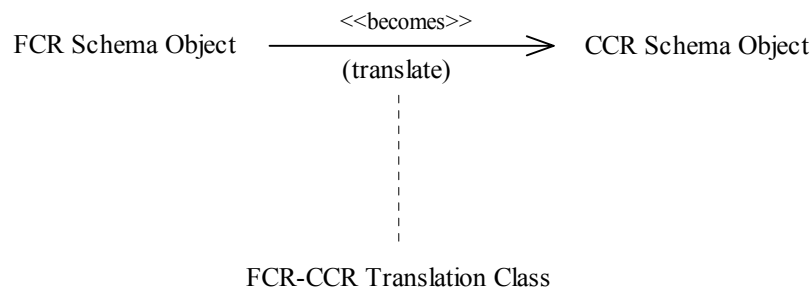


Figure VII-17. FCR Schema Object to CCR Schema Object Translation

From our continuing example, the GroundCombatVehicle_View3__ArmoredFightingVehicle Translation class associated with the destination ArmoredFightingVehicle CCR Schema, would be used to convert the GroundCombatVehicle_View3 FCR Schema object created from the received GroundCombatVehicle_View1 FCR Schema object to its behaviorally equivalent destination

object representation. Figure VII-18 illustrates application of this class's `translate(groundCombatVehicle_View3 : GroundCombatVehicle_View3) : ArmoredFightingVehicle` method on the received `GroundCombatVehicle_View3` FCR Schema instance to produce an instance of the `ArmoredFightingVehicle` CCR Schema.

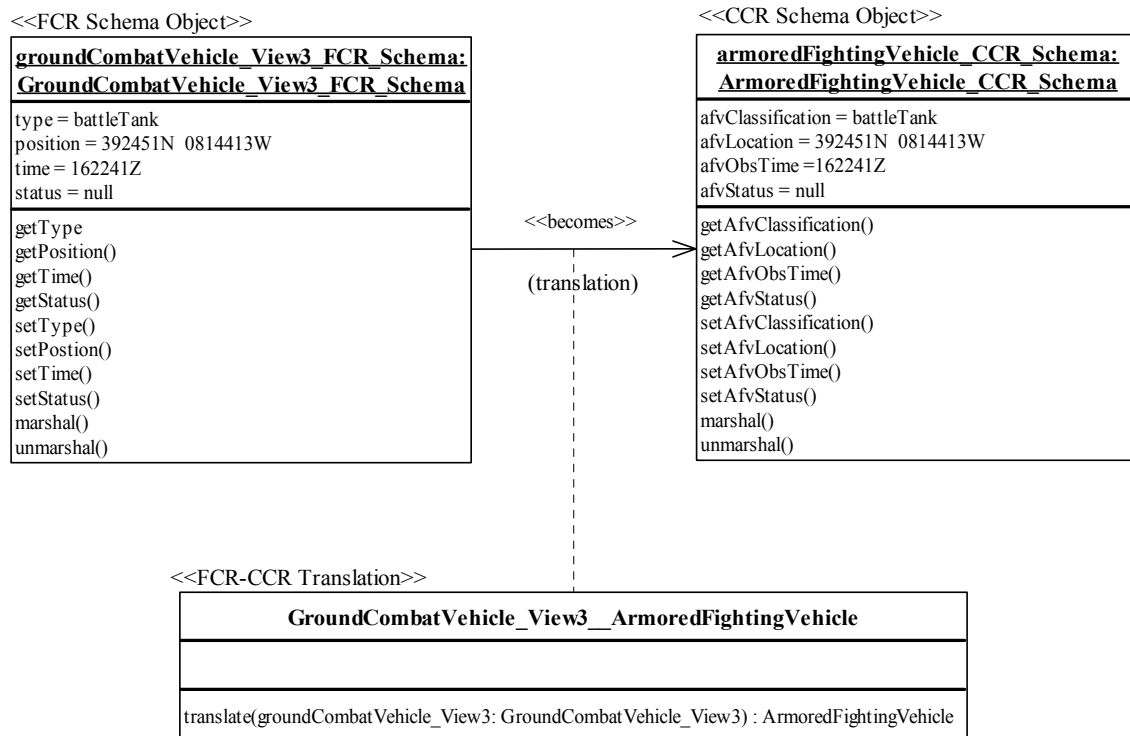


Figure VII-18. Translating from Intermediate to Destination Object Representation

d. Converting From Destination Object Representation to Destination XML Document Representation

The last step in converting from an intermediate to a destination model of a real-world entity involves the conversion from an object representation of the destination model of the received real-world entity to an XML document representation. Conversion from a destination object representation to an XML instance document representation of the exported information is accomplished by use of the `marshal` method generated with the CCR Schema defining the destination object representation. The destination translator uses this CCR Schema object's `marshal` method to convert the object representation to its equivalent XML instance document representation. Figure VII-19 illustrates the process for converting the CCR Schema object to its equivalent

XML instance document representation and the relationship between the CCR Schema object's defining CCR Schema and the XML instance document's governing XML Schema.

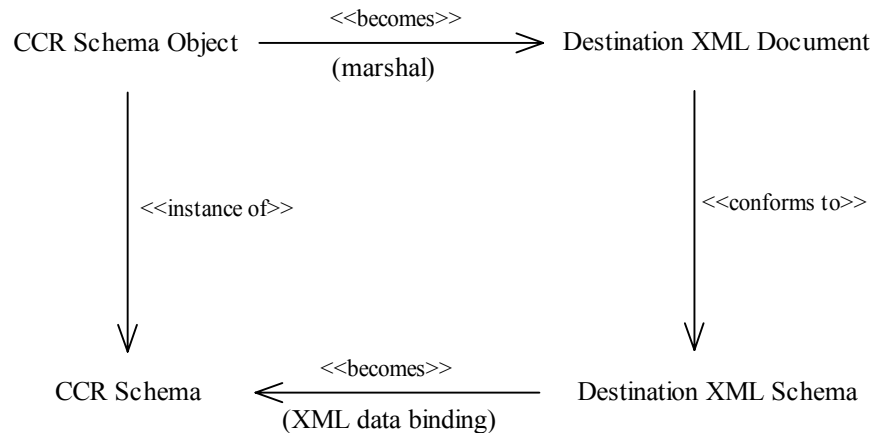


Figure VII-19. Process for Converting CCR Schema Object to its Equivalent XML Instance Document

From our continuing example, Figure VII-20 depicts the destination XML Schema used to generate the selected ArmoredFightingVehicle CCR Schema. The destination model translator uses the marshal method from this class to convert the destination CCR Schema instance to an equivalent destination XML instance document. This destination XML instance document is then forwarded to the destination system application for action. Figure VII-21 illustrates the conversion from an object representation of our ArmoredFightingVehicle object to its equivalent XML instance document representation, the contents of which are displayed in Figure VII-22.

D. TRANSLATOR SUMMARY

The purpose of the translator presented in this chapter is to resolve heterogeneities in state and behavioral information shared among systems in order to enable their interoperation. These heterogeneities can be categorized as either a difference in view or differences in representation of a view of the real-world entities whose state and behavior are shared among systems. Under the Object-Oriented Method for Interoperability (OOMI), an interoperability engineer captures these differences in

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://nps.navy.mil/cs/oomi/systemD"
xmlns:systemD="http://nps.navy.mil/cs/oomi/systemD/armoredFightingVehicle"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xsd:element name="armoredFightingVehicle">
    <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation>An armoredFightingVehicle msgtype provides System D model of ground
          combat vehicle real-world entity for the example in Figure III-3.</xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="afvClassification" type="systemD:afvClassificationType"/>
        <xsd:element ref="systemD:afvLocation"/>
        <xsd:element ref="systemD:afvObsTime"/>
        <xsd:element name="afvStatus" type="systemD:afvStatusType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="afvClassificationType">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="5"/>
      <xsd:maxLength value="14"/>
      <xsd:enumeration value="battleTank"/>
      <xsd:enumeration value="rocketLauncher"/>
      <xsd:enumeration value="truck"/>
      <xsd:enumeration value="unknown"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="afvLocation">
    <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation/>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="latitude" type="systemD:latitudeType"/>
        <xsd:element name="longitude" type="systemD:longitudeType"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="afvObsTime">
    <xsd:complexType>
      <xsd:annotation>
        <xsd:documentation>The day of a month and timekeeping in hours and minutes of a
          calendar day, using the 24-hour clock system referenced to Greenwich Mean Time
          (GMT).</xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="day" type="systemD:dayType"/>
        <xsd:element name="hourTime" type="systemD:hourTimeType"/>
        <xsd:element name="minuteTime" type="systemD:minuteTimeType"/>
        <xsd:element name="stdTimeZone" type="systemD:stdTimeZoneType"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="afvStatusType">
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="7"/>
      <xsd:maxLength value="11"/>
      <xsd:enumeration value="operational"/>
      <xsd:enumeration value="damaged"/>
      <xsd:enumeration value="destroyed"/>
    </xsd:restriction>
  </xsd:simpleType>
  .
  .
  .
</xsd:schema>

```

Figure VII-20. Destination XML Schema "armoredFightingVehicle.xsd"

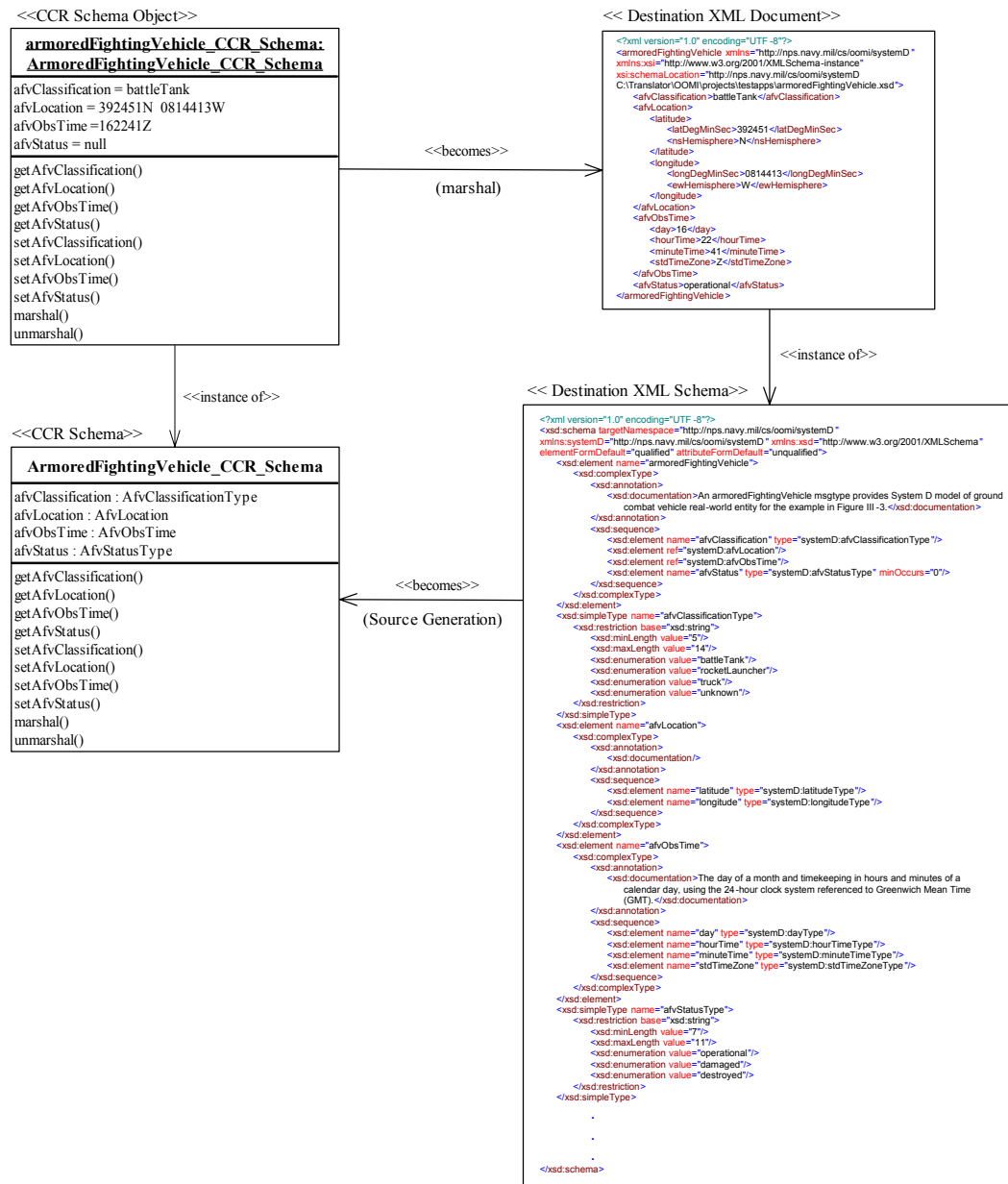


Figure VII-21. Conversion From Destination Object Representation to Destination XML Instance Document Representation

view and representation in a Federation Interoperability Object Model (FIOM) created for a specified federation of systems prior to runtime. Then at runtime, information contained in the FIOM is used to resolve system modeling differences.

Differences in view are resolved through exploitation of the commonalities among systems of different views of the same real-world entity. Differences in view

representation are resolved by use of translations defined by the interoperability engineer prior to runtime and stored in the FIOM for each component system model of the real-world entity. In order to minimize the number of required translations, the OOMI translator uses a two-step process, involving the use of an intermediate model for each real-world entity involved in system interoperation. In the first step the source model of a real-world entity is translated to an equivalent intermediate model of that entity. In the second step, the intermediate model of the real-world entity is translated to a behaviorally equivalent model suitable for use by the destination system.

```
<?xml version="1.0" encoding="UTF-8"?>
<armoredFightingVehicle xmlns="http://nps.navy.mil/cs/oomi/systemD"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://nps.navy.mil/cs/oomi/systemD
C:\Translator\OOMI\projects\testapps\armoredFightingVehicle.xsd">
  <afvClassification>battleTank</afvClassification>
  <afvLocation>
    <latitude>
      <latDegMinSec>392451</latDegMinSec>
      <nsHemisphere>N</nsHemisphere>
    </latitude>
    <longitude>
      <longDegMinSec>0814413</longDegMinSec>
      <ewHemisphere>W</ewHemisphere>
    </longitude>
  </afvLocation>
  <afvObsTime>
    <day>16</day>
    <hourTime>22</hourTime>
    <minuteTime>41</minuteTime>
    <stdTimeZone>Z</stdTimeZone>
  </afvObsTime>
  <afvStatus>operational</afvStatus>
</armoredFightingVehicle>
```

Figure VII-22. Destination XML Instance Document "armoredFightingVehicle.xml"

Source to intermediate model translation first involves conversion of the information exported by a source system from an XML instance document representation to an object representation of that information, captured in the OOMI as a Component Class Representation (CCR) Schema object. The next step involves conversion from the source object representation to the corresponding intermediate object representation, depicted as a Federation Class Representation (FCR) Schema object under the OOMI. The final optional step entails converting the intermediate object representation to an intermediate XML instance document representation if the federation architecture uses

XML for transferring information among systems. Figure VII-23 illustrates the source to intermediate model translation process implemented by the OOMI translator.

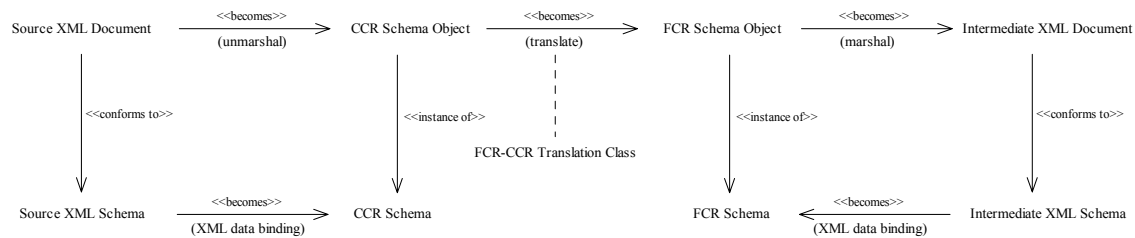


Figure VII-23. Source to Intermediate Model Translation

Translation from an intermediate to a destination model of a real-world entity defining an interoperation requires resolution of potential differences in view as well as possible differences in representation between intermediate and destination models. Unless the source and destination systems have the same view of the real-world entity being modeled, the intermediate model will have a different view of the modeled real-world entity than the destination system. Intermediate to destination model translation therefore first requires resolution of the differences in view between the intermediate and destination models and then resolution of differences in view representation.

Converting from an intermediate to a destination model of a real-world entity involves four steps. First, if the translator receives the intermediate model in the form of an XML instance document, then it must first convert the received XML instance document to its equivalent object representation, captured as an FCR Schema object in the OOMI. Second, the translator must resolve differences in view between the intermediate and destination models. Third, the translator must resolve differences in representation for a specified view, producing a CCR Schema object reflective of the destination model of the real-world entity. Fourth, the destination model must be converted from an object representation to an equivalent XML instance document representation for forwarding to the destination system application. Figure VII-24 depicts the intermediate to destination model translation process of the OOMI translator.

The OOMI translator presented in this chapter offers an improvement in the methodology used by systems in Chapter II representing the current state of the practice

toward achieving system interoperability. This improvement is provided in three areas. First, the OOMI translation methodology utilizes a two-step process that requires the definition of a maximum of $2n$ translations to resolve representational differences among a federation of n systems. This is a reduction from the $n(n-1)$ translations required using point-to-point system interfaces when the number of involved systems exceeds three. Second, the runtime translation process is fully automated requiring no operator intervention once an FIOM is constructed for a specified system federation. Third, the OOMI translator provides the capability to log translation failures for input to the requirements process for future component system modification.

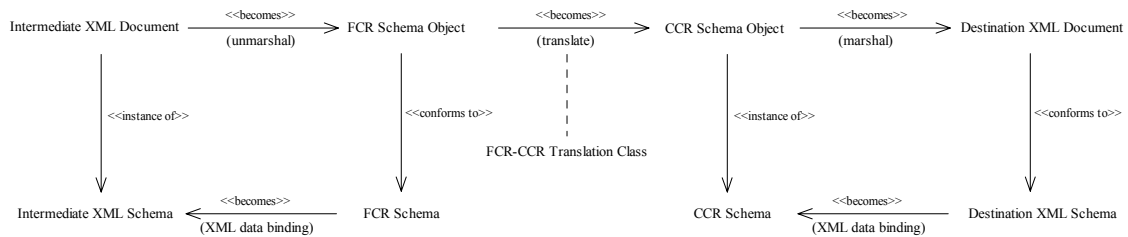


Figure VII-24. Intermediate to Destination Model Translation

The OOMI translator prototype implemented by [Lee02] demonstrates the use of a two-step translation process in automating the conversion between models of information shared among systems in a sample federation. The capability for translation failure logging is not provided in the initial prototype, but is planned for future implementations.

VIII. CONCLUSION AND RECOMMENDATIONS FOR FUTURE RESEARCH

A. REVIEW OF CRITERIA USED FOR EVALUATING INTEROPERABILITY APPROACHES AND LIMITATIONS SEEN IN CURRENT SYSTEMS

In Chapter II eight criteria for evaluating approaches for achieving interoperability among heterogeneous systems were identified. Those criteria are listed below and explained in Section II.B. The eight criteria used are:

- Types of heterogeneity addressed
- Capability for application of computer aid for model correlation
- Required knowledge of remote operations
- Required modification to existing system
- Translation methodology used
- Capability for application of computer aid for translation development
- Support for Federation Extensibility
- Information exchange versus joint task execution

Six of the most pertinent existing approaches for achieving interoperability among heterogeneous systems were evaluated against these criteria. From that evaluation, the following limitations were seen in some or all of the existing approaches.

- None of the systems evaluated provide a means for resolving the complete spectrum of modeling differences found among heterogeneous systems.
- Only one of the existing approaches provide assistance in determining when different system models refer to the same entity from the problem domain.
- All of the current approaches require a requesting system to conform to a provider's model of requested state or behavior information, requiring modification to the requestor if not already conformant.
- Most of the evaluated approaches utilize a point-to-point conversion process for resolving modeling differences among systems vice a two-step process using an intermediate model, resulting in a greater number of translations to be defined when the number of systems to be integrated exceeds three.
- Most of the approaches provide little or no support to the development of the translations required to resolve modeling differences among systems.
- Most approaches are concerned only with the resolution of modeling differences for information exchanged among systems and do not provide the capability for resolving possible differences in the signatures used to access the behavior of corresponding methods on different systems.

In response to these limitations and in answer to the research question raised in Section I.C, the Object-Oriented Method for Interoperability (OOMI) introduced in Chapter IV was developed to provide a means for resolving the differences in data models among heterogeneous systems that have hampered the quest for system interoperability. An evaluation of the OOMI's success in this endeavor is provided next.

B. EVALUATION OF OBJECT-ORIENTED METHOD FOR INTEROPERABILITY AGAINST INTEROPERABILITY COMPARISON CRITERIA

In this section the OOMI is evaluated against the Section II.B criteria used for comparing interoperability approaches. As part of the discussion, a look at how the OOMI addresses the limitations seen in the previous interoperability approaches is provided. The results of that evaluation are summarized in Table VIII-1 and discussed in the following paragraphs.

1. Types of Heterogeneity Addressed

The OOMI provides the interoperability engineer with the means for addressing each of the types of heterogeneity defined in Section II.A.2. Under the OOMI, a Federation Entity (FE) is defined for each real-world entity whose state and behavior are shared among systems in a federation. As described in Section IV.C.1.b(1), for each FE one or more Federation Entity Views (FEVs) are used to distinguish differences in *what* real-world-entity characteristics are modeled by different systems due to heterogeneities in scope, level of abstraction, or temporal validity. These differences are captured by variations in the number and composition of the attributes and operations used for modeling the same real-world entity on different systems. Commonalities in the attributes and operations used to define an FEV are used to construct an inheritance hierarchy for the FEV relating the models that can be used to determine when the information contained in one system's view of an entity is suitable for use by another.

In addition to differences in what characteristics are chosen to model a real-world entity, there may be variations in *how* these characteristics are represented on different component systems. These variations may be due to heterogeneities of hardware and operating systems, organizational models, structure, presentation, and meaning found on the different systems. In order to capture these differences, the OOMI provides two mechanisms to denote the possible alternative representations of an entity's view. The

first mechanism, the *Federation Class Representation (FCR)*, is used to reflect the “standard” (as defined by the interoperability engineer) representation used by the federation for an entity’s view. The second mechanism, the *Component Class Representation (CCR)*, is used to capture a component system’s representation of an FEV.

Table VIII-1. Evaluation of OOMI Support for Resolution of Modeling Differences

Evaluation Criteria	OOMI
Types of Heterogeneity Addressed	Hardware and Operating System; Organizational Models; Structure; Presentation; Meaning; Scope; Level of Abstraction; Temporal Validity;
Capability for Application of Computer-Aid for Model Correlation?	Yes. Syntactic and Semantic correlation algorithm.
Knowledge of Remote System Methods Required?	No. Correlation algorithm will assist Interoperability Engineer in finding remote system method corresponding to local (client) invocation.
Modification to Existing System Required?	No. Translator implemented as middleware or using wrapper-based approach.
Translation Methodology?	Two-step using intermediate representation.
Capability for Application of Computer-Aid for Translation Development?	Computer-aided generation of translation skeleton; library maintained for functional transformation reuse.
Support for Federation Extensibility	Partial support. Additions or changes to existing FCR Schema attribute and operation representations may affect existing translations used to resolve representational differences between CCR and FCR Schemas.
Information Exchange vs. Joint Task Execution	Both information exchange and joint task execution.

Differences between component representations are resolved by means of a two-step translation process whereby a source CCR is first converted to its equivalent FCR and then to the corresponding destination CCR. *Translations* are defined for each FEV to enable conversion between each CCR and the corresponding FCR. Section IV.C.1.b(3) details the means for resolving FEV representation differences.

As described in Chapter VII, the OOMI translator uses information from the FEV's FCR Schema Inheritance Hierarchy and *FCR-CCR Translations* to resolve heterogeneities among federation systems. The translator uses the FEV's FCR Schema Inheritance Hierarchy to resolve differences in view among systems, as caused by heterogeneities of scope, level of abstraction, and temporal validity. The translator uses an FCR-CCR Translation associated with each component representation of a view (CCR) to resolve heterogeneities of hardware and operating systems, organizational models, structure, presentation, and meaning.

2. Capability for Application of Computer Aid for Model Correlation

During construction of a Federation Interoperability Object Model (FIOM) for a specified federation of systems, correspondences between information and operations exported from or imported to the component systems must be identified in order to enable system interoperability. The OOMI Integrated Development Environment (IDE) provides computer-aid for identifying these correspondences.

As described in Section VI.B, the correlation methodology implemented for the OOMI IDE is used to assist the interoperability engineer in adding a CCR to the FIOM during the *Register CCR* phase of IDE operation. Assistance is provided to the interoperability engineer in terms of computer aid for finding the FE corresponding to the same real-world entity modeled by the CCR being registered. The OOMI IDE correlation methodology uses a two-phased approach for establishing this correspondence.

In the first phase, semantic information in the form of keywords taken from descriptions of the component and federation models of the real-world entity is used to establish the correspondence. In the second phase, details about the structure and composition of the attributes and operations used for the component and federation models of a real-world entity are used to correlate the models. Potential correspondence

between a CCR and an FCR is given in terms of a score for both the semantic and syntactic phases of the correlation effort. Comparison of scores among potential FCR matches will direct the interoperability engineer toward the most likely match for a CCR. However, final determination of CCR-FCR correlation requires a one-to-one correspondence between the attribute and operation sets of a potential CCR-FCR match as discussed in Section V.C.3.b. This determination is the responsibility of the interoperability engineer and is not automated in the OOMI IDE.

3. Required Knowledge of Remote Operations

Under the OOMI, definition of an intermediate model of the real-world entities involved in the interoperation between systems as described in Section IV.C.1.b(2) and the use of the two-step translation methodology described in Section VII.C mean that one system does not have to prior knowledge of another in order to exchange information or execute the other's tasks.

For information exchange the source system provides the exported information in the form of a set of attributes or objects of a producer class in the native format of the producer. The exported information is first converted into a corresponding intermediate model by an OOMI source model translator using a source-to-intermediate model translation captured as part of the FIOM during federation design. Then, in a second step, executed by a corresponding OOMI destination model translator, the intermediate model of the exported information is converted into the destination consumer system model. Neither the source nor destination systems require any knowledge of the information model used by the other system. Each is concerned only with the conversion between component and intermediate models using translations defined during federation design.

For joint task execution, a client system provides an operation name and a set of parameter values for a desired operation in the native format of the client. The parameters may be attributes, operations, or objects of a client class. The OOMI source model translator first converts the operation name and parameter values into a corresponding intermediate model using a source-to-intermediate model translation captured in the FIOM. Then, the OOMI destination model translator for a system containing an implementation of a behaviorally equivalent operation converts the

intermediate model of the operation name and parameter values to the corresponding model recognized by the destination server system. Again, neither the source nor destination systems require any knowledge of the information model used by the other system.

The information required to effect these translations is captured as part of the FIOM during federation design. Correlation software is used to help an interoperability engineer locate other component systems containing a model of the same real-world entities exported or imported by a particular system. This information is used in constructing a federation model of the real-world entities involved in the system interoperation (the FIOM). Then, at run-time, an OOMI translator accesses the information contained in the FIOM to resolve differences in view of the component system models and to effect the translation between component and standard representations of a view.

4. Required Modification to Existing System

The use of an intermediate model of the real-world entities involved in the interoperation between systems and the use of the two-step translation methodology described in Section VIII.B.3, together with a wrapper or middleware based implementation of the OOMI translator, eliminates the requirement for existing system modification in order to resolve heterogeneities among system models.

In a wrapper-based approach, the translator is implemented as a software intermediary that logically envelops a component system. The wrapper functions to intercept incoming and outgoing information from the wrapped system and convert it between component and intermediate models. In a system federation where the translator is implemented as part of middleware in a hub-and-spoke architecture, the translator serves the same function as in the wrapper-based approach. The primary difference between the architectures lies in the number of translators required and in the location of the translator(s). In a hub-and-spoke architecture a single source and destination model translator is typically implemented on a separate platform between the source and destination systems. In a wrapper-based architecture, separate translators are required for each system in the federation and will commonly reside on the same hardware platform as the component system.

As a result of the two-step translation methodology, no modifications to the component systems are required to resolve differences among system models. The translator accomplishes any translations required between models. Modifications to the component systems are required only if a change to the external interface is required to expose information or operations available internally in a component system or to take advantage of information or operations exposed by other systems.

5. Translation Methodology

As pointed out in Section VII.C the OOMI translator uses a two-step process, involving the use of an intermediate representation, to reduce the number of required translations. In the first step, an object conforming to the source model of a real-world entity, captured as a CCR instance, is translated to an equivalent object conforming to the intermediate model of that entity in the form of an FCR instance. In the second step, the intermediate model object of the real-world entity is translated to a behaviorally equivalent model suitable for use by the destination system. This pair-wise approach to resolving representational differences between systems reduces the number of required translations from $n(n-1)$ to $2(n)$ for a federation of n systems.

6. Capability for Application of Computer Aid for Translation Development

As discussed in Section V.D.2.c, given corresponding federation and component models of a real-world entity whose information and operations are shared among systems, the OOMI IDE assists the interoperability engineer with defining the translations required to resolve representational differences between the models. The OOMI IDE provides computer aid to the interoperability engineer for defining a translation in two areas. The first area involves exploiting user-identified correspondences between component and federation models to provide a framework for translation definition. The second area involves creation and maintenance of a library of pre-defined translation definitions for insertion into this translation framework.

7. Support for Federation Extensibility

Use of object-oriented design principles in the development of the FIOM supports creation of a model for achieving interoperability among heterogeneous systems that is extensible. When adding a new component system to the federation, an evaluation is

made to determine if the real-world entities modeled by the component system are already represented in the FIOM. If not, a new federation entity, complete with views and representations of those views, is added to the federation to which the component model of the entity will be included. If such an FE already exists in the FIOM, then a determination is made as to whether there exists a view in the FE's FCR Schema Inheritance Hierarchy whose FCR Schema exhibits a one-to-one correspondence with the attribute and operation set of the component system model of the real-world entity. If so, the system designer simply adds a new CCR reflecting the component system model of the entity to the appropriate view. If such a view does not exist, a new FEV is added for that entity to which the CCR will be included, and relationships established between the new FEV and existing views in the entity's inheritance hierarchy. Addition of federation entities, entity views, and representation of those views is accomplished without affecting existing model components or relationships and therefore without affecting existing translations and existing interoperability data path implementations.

Modifying existing FCRs to change the representation of the FCR Schema attributes and operations presents a potential problem. No FCR versioning support is provided in the FIOM. Using inheritance to support versioning is complicated by the use of the FCR Schema Inheritance Hierarchy to capture relationships between different views of real-world entity. Using inheritance at the FEV/FCR level would introduce problems associated with multiple inheritance. Therefore, implementation of versioning support through inheritance would have to be implemented at the FE or FIOM level.

In summary, additions to the federation that do not require modification to existing FCR Schema attribute and operation representations can be made without impacting interoperation of the original system federation. Additions or changes that do require modification to existing FCR Schema attribute and operation representations may affect existing translations used to resolve representational differences between CCR and FCR Schemas. Therefore, the OOMI is considered to provide partial support for federation extensibility.

8. Information Exchange Versus Joint Task Execution

The OOMI provides the capability for resolving system heterogeneities during both information exchange and joint task execution. As discussed in Section VII.A and

reiterated in Section VIII.B.3, for information exchange the OOMI translator(s) convert exported information from the source system model to an equivalent model required by the destination system in a two-step translation process. For joint task execution, the OOMI translator(s) similarly convert the name and parameters used to request a remote system operation from the client system model to the equivalent model required by the remote server system.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

As an extension of the work provided in this dissertation, six areas for future research are recommended. First, an evaluation of the efficiency and effectiveness of the OOMI in creating an interoperable federation from a sample group of independently developed systems is recommended. Second, investigation of potential enhancements to the syntactic and semantic correlation methodology used to compare component and federation models of the real-world entities that define the interoperation among systems should be considered. Third, further research is recommended to determine the means where information from two or more different producers can be combined to create a new view satisfying a consumer's requirements. Fourth, an investigation of areas where the correlation methodology used for comparing component and federation models of shared entities can be expanded to other aspects of the FIOM construction process should be conducted. Fifth, extension of the OOMI IDE to operate as a distributed network application, thereby serving as an enabler for the broader application of the FIOM in providing an interoperability framework across a domain or domain segment, is suggested. Finally, an investigation of the application of the methodology described in this dissertation to the real-time system domain should be considered.

1. Evaluation of Efficiency and Effectiveness of OOMI in Creating an Interoperable Federation of Systems

Key areas implemented in the initial OOMI IDE prototype include the 1) User Interface, 2) Component Model Correlator, 3) Translation Generator, 4) FIOM Database and 5) portions of the Federation Entity Manager. In addition, implementation of the core heterogeneity resolution functionality of the OOMI translator has been completed. Completion of the OOMI IDE prototype implementation and integration of the OOMI translator core functionality into either a wrapper-based or middleware application is

required in order to evaluate the efficiency and effectiveness of the OOMI in creating an interoperable federation from a sample group of autonomously developed component systems. Such an evaluation should include measurement of the time and effort required to create a federation of interoperable systems using the OOMI, an assessment of the OOMI's capability for resolving the complete list of heterogeneities specified in Section II.A.2, and an appraisal of the correlation methodology used for FIOM construction.

While Section VIII.B detailed the methodology used by the OOMI for resolving each of the types of heterogeneity specified in Section II.A.2, a complete assessment of the OOMI's capability for heterogeneity resolution requires application of the methodology for actual system integration. Such an assessment would require integration of systems exhibiting the complete spectrum of heterogeneities specified in Section II.A.2.

The primary metrics to be used for an assessment of the correlation methodology used for FIOM construction are a determination of the precision and recall values attained when correlating component and federation models of the real-world entities involved in system interoperation. As discussed in Sections VI.B.2.a and VI.B.2.b the interoperability engineer can set a threshold for display of candidate matches returned by either the semantic or syntactic correlation methodologies or by a combination of the two methods. Values for precision and recall should be computed for each of these possible correlation alternatives, using a range of threshold values as input.

2. Enhancements to Correlation Methodology

a. Semantic Correlation Methodology

The semantic correlation methodology used in the initial prototype OOMI IDE is limited to providing a percent of keywords in the syntax component of a CCR matching keywords in an FCR's syntax component. Enhancement of the semantic correlation methodology to include synonyms to the CCR keyword as matches should increase precision and recall values returned by the semantic correlator. Further improvement should also result from inclusion of a concept search feature such as provided with the Personal Librarian (PL) tool found in MITRE's DELTA tool [BFH+95].

b. Syntactic Correlation Methodology

Improvements in the syntactic correlation methodology used in the OOMI IDE can be made in two areas. First, an increase in the number of data schema related elements included in the discriminator vectors used for neural network training and correlation determination should improve correlator performance. Second, inclusion of data content as well as schema information in constructing the discriminator vectors for both component and federation models will add a limited semantic discrimination capability to the neural networks used for model correlation.

Additional schema related elements that could be incorporated in the discriminator vectors include:

- data precision
- whether nullable variables are allowed
- number of possible values for enumerable types
- whether default value is provided

While not a complete list of schema related information that might be available, the above elements should be easily extractable from the schema used to define a real-world entity's structure and content.

In the original SEMINT tool, the authors used data content as well as schema related information for constructing the discriminator vectors [LC00]. The addition of statistical information on the actual contents of the XML documents being produced and consumed by a system could provide a limited perspective on the semantics of an entity's attributes or operations. Additional data content level information such as the maximum and minimum values observed, and statistics on such observed values to include the average, standard deviation, and coefficient of variance could potentially help discriminate between elements that have the same type and composition. For example, two elements could be modeled as six digit integers, but each provide a different usage; i.e., one element's values are always in the range of 000000 to 235959 indicating a possible time-related quantity, whereas the other element's values range from 000000 to 999999 signifying an unrestricted numeric quantity. Capturing data content information on these two elements would enable the neural network to distinguish their different uses.

3. Enabling Join Operations for Federation Entity View Definition

In order to avoid difficulties with multiple inheritance in the construction of the FCR Schema Inheritance Hierarchy, join operations where information from two or more different producers can be combined to create a new view satisfying a consumer's requirements are not permitted. Methods for enabling such join operations while minimizing the difficulties associated with multiple inheritance are identified as an area for future research.

4. Expansion of Correlation Methodology Application During FIOM Construction

a. Application of Correlation Methodology to Mapping of Corresponding Attributes and Operations during Translation Generation.

Using correspondences between component and federation models identified by the user, the OOMI IDE *Translation Generator* provides a framework for translation definition. The user is presented with a graphical representation of the CCR and FCR Schemas from a Federation Entity View (FEV) and then given the capability to match attributes and operations between the two representations of that view via a “click-to-select” procedure. Applying the correlation methodology defined in Section VI.B.2 to the attribute and operation mapping process might prove beneficial in reducing the time and effort required for translation creation. Attribute and operation correlation techniques could be used to provide a candidate mapping for interoperability engineer approval.

b. Application of Correlation Methodology and Behavioral Equivalence Determination Algorithms to Modification of FIOM to Provide Required One-To-One Correspondence Between CCR and FCR Schemas During CCR Registration.

If during FIOM construction, an existing FCR Schema cannot be found whose attribute and operation sets provide a one-to-one correspondence with the CCR Schema attributes and operation sets for the CCR being registered and whose operations are behaviorally equivalent, then such an FCR Schema must be added to the FIOM. The conforming FCR Schema can be added either through addition of a new FE with constituent FCR and defining FCR Schema or through generalization or specialization of an existing FCR Schema. When creating a new FCR Schema through generalization of

an existing FCR Schema, the interoperability engineer must identify the attributes and operations in the selected existing FCR Schema that correspond to the attributes and operations of the CCR Schema being registered. In addition he must verify behavioral equivalence between the two schemas' operations.

A similar situation exists when creating a corresponding FCR Schema through extension of an existing FCR Schema. The interoperability engineer selects the FCR Schema that most closely corresponds to the CCR Schema being registered. He then must provide a mapping of the attributes and operations in the selected existing FCR Schema that correspond to the attributes and operations of the CCR Schema being registered. In addition, he must define the FCR Schema attributes and operations for the new FEV that correspond to the unmapped CCR Schema attributes and operations using information contained in the Federation Ontology.

The scenarios outlined above present three opportunities for future research. First, investigation of how the correlation methodology defined in Section VI.C.2 could be applied to the process of mapping CCR and FCR Schema attributes and operations needed for creating an FCR Schema corresponding to the CCR Schema being registered. Second, an investigation of how the correlation methodology could be applied to locating information in the Federation Ontology to be used during the addition of new attribute and operations to the FCR Schema. The third opportunity for potential research is an investigation of possible algorithms that can help the interoperability engineer determine whether corresponding operations between an FCR and CCR Schema are behaviorally equivalent.

5. Extending OOMI IDE to Operate as a Distributed Network Application

Another area for future research would be to develop a session-layer networking protocol that allows the OOMI IDE to be deployed as a distributed network application accessible via the Internet. The protocol would serve as an International Standards Organization (ISO) Open Systems Interconnection (OSI) session layer protocol that would be implemented on top of transport and internet protocols such as TCP/IP [Fei99]. By extending the OOMI IDE to operate as a distributed network application, the

advantages gained through the creation of an FIOM for a specified federation of systems can be extended to a domain or higher level.

The key to success in sharing information between independently developed systems is to develop a common, flexible framework under which all entities can resolve their differences. The framework should provide a common model of the information to be shared in order to minimize the number of translations required to resolve differences among systems. The framework should be flexible in order to allow modification to the model without impacting existing applications' use of the model. The FIOM provides the framework required to allow dissimilar systems to communicate. In this sense the FIOM provides a function similar to that envisioned by the DII/COE XML Namespace Registry [DII01]. However, information included with the FIOM provides more than just the name and type information provided in the DII/COE XML Namespace Registry for the entities shared among systems. It provides sufficient information for both federation and component models of those shared entities to resolve the full spectrum of heterogeneities discussed in Section II.A.2.

Development of a distributed, network-based OOMI IDE application serves as an enabler for defining an FIOM across a wider problem environment. By making the OOMI IDE available across the Internet, interoperability engineers from an entire domain or domain segment could participate in construction of the Federation Entities to be shared among systems. Using functionality proposed for the Federation Ontology Manager discussed in Section V.D.4, an interoperability engineer could select the appropriate federation representation for a real-world entity of interest or nominate a change to the Federation Ontology if such an entity is not found. The resulting FIOM would provide a common language for communication within the domain or domain segment [Law01].

6. Resolution of Modeling Differences in Real-Time Systems

In addition to problems associated with differences in modeling of heterogeneous systems in a federation, achieving interoperability among real-time and non-real-time systems presents added challenges. Under the OOMI differences in *what* characteristics heterogeneous systems use to model a real-world entity and differences in *how* those characteristics are represented are resolved. In addition, real-time systems are concerned

with coordinating *when* the information or operations characterizing such real-world entities are shared among federation systems. In order for such real-time systems to interoperate, they not only require resolution of modeling differences among systems but also require that the information and operations shared among systems be presented in an expected sequence and that specified timing constraints be met.

Systems such as Luqi's Computer-Aided Prototyping System (CAPS) [LB88] can be used to create a system federation that addresses the timing concerns associated with real-time systems. For example, the Prototype System Description Language (PSDL) used in CAPS provides mechanisms for specifying timing constraints required for real-time operation such as Maximum Execution Time (MET), Period, and Finish Within (FW) for periodic events and MET, Minimum Calling Period (MCP) and Maximum Response Time (MRT) for sporadic events [LBY88]. However, such systems typically assume a homogeneous model of the problem environment and do not take modeling differences associated with heterogeneous systems into consideration.

Combining the capabilities of a real-time modeling system such as CAPS and the heterogeneous system modeling difference resolution capabilities of the OOMI to support the interoperation of real-time and non-real-time systems is identified as an area of interest for future research. Possible alternatives for accomplishing such an objective include incorporating timing constraint resolution mechanisms in the OOMI, including the heterogeneous system modeling difference resolution capability into a CAPS or similar system, or the use of a layered approach whereby modeling difference resolution is accomplished using the OOMI and timing constraints are imposed by a CAPS using PSDL or a similar real-time control mechanism.

D. CONCLUDING REMARKS

In the Chapter I introduction to this dissertation, I presented the following hypothesis in response to the question of whether computer aid could be applied to the problem of resolving data modeling differences among systems targeted for integration in order to enable system interoperability.

By using a model-based approach, a computer-aided methodology can be provided to aid in the resolution of data modeling differences among systems targeted for integration in order to enable system interoperability.

As evidenced by the Section VIII.B evaluation of the Object-Oriented Method for Interoperability (OOMI) presented in this dissertation against the interoperability comparison criteria specified in Section II.B, the above hypothesis is affirmed. The OOMI provides such a methodology.

APPENDIX A: MODIFYING FIOM TO PROVIDE REQUIRED CORRESPONDENCE BETWEEN CCR AND FCR SCHEMAS DURING CCR REGISTRATION

A. ADDING NEW FEDERATION ENTITY (FE) TO FEDERATION INTEROPERABILITY OBJECT MODEL (FIOM)

If during Component Class Representation (CCR) registration there is no Federation Entity (FE) in the Federation Interoperability Object Model (FIOM) that corresponds to the same real-world entity as the CCR being registered, then the interoperability engineer must define a new FE for that real-world entity. Such could potentially be the case when adding the first component system model of a real-world entity to the FIOM. The newly defined FE will include a Federation Entity View (FEV) with Federation Class Representation (FCR) Schema containing attribute and operation sets that exhibit a one-to-one correspondence with the CCR Schema attribute and operation sets. That is, a function $f: FCR \rightarrow CCR$ must exist mapping the FCR Schema attribute and operation sets ($FCR(A\epsilon, \Omega\epsilon)$) to CCR attribute and operation sets ($CCR(A\epsilon, \Omega\epsilon)$) that is one-to-one and onto and whose operations are behaviorally equivalent. New FCR Syntax and FCR Semantics components will be automatically generated from this FCR Schema and included with the new FEV. The CCR, with component CCR Schema, CCR Syntax, and CCR Semantics, will also be included with the new FEV and an association established between the new FEV's FCR Schema and the schema for the CCR being registered. The Interoperability Engineer is responsible for definition of the FCR Schema attributes and operations corresponding to the CCR Schema attribute and operation sets.

In order to illustrate the creation of a new FE when adding the first component system model of a real-world entity to the FIOM, I will show the addition of the first component system model of the example ground combat vehicle real-world entity seen in Figure IV-3, System A's armoredCombatVehicle, to the FIOM. As indicated by the class correlation process and confirmed by the interoperability engineer, initially the FIOM will not contain an FE corresponding to the ArmoredCombatVehicle_CCR_Schema

depicted in Figure A-1. Therefore, the interoperability engineer must define a new FE to represent the real-world entity modeled by System A's armoredCombatVehicle.

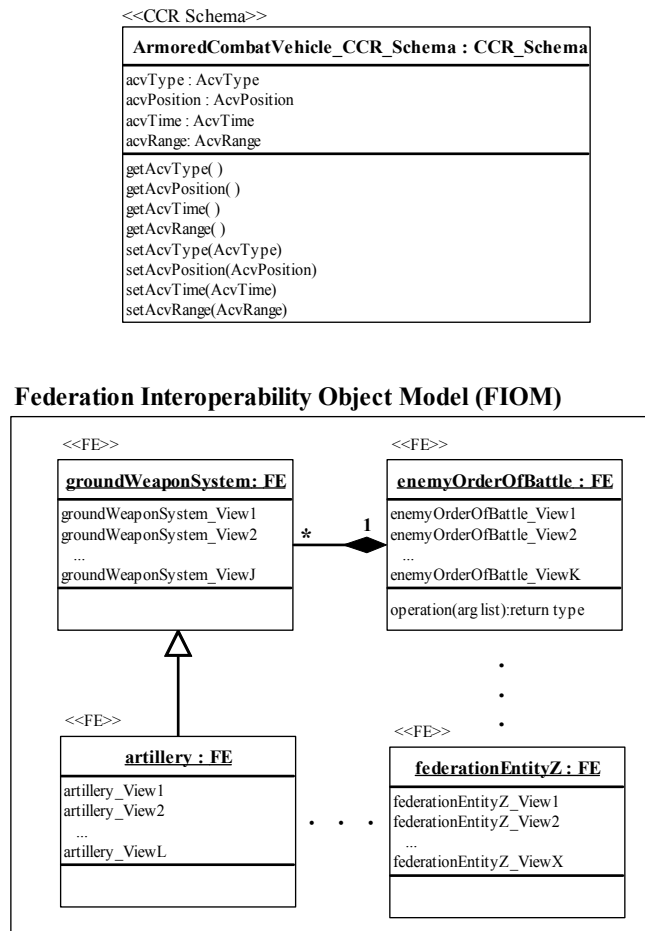


Figure A-1. No FE in FIOM Corresponding to CCR Being Registered

As part of this FE, the interoperability engineer must define an FEV that captures the perspective of the real-world entity depicted by System A's armoredCombatVehicle model. As discussed in Section IV.C.1.b(2), System A's perspective of the real-world entity is defined by the attribute and operation sets contained in its ArmoredCombatVehicle_CCR_Schema. The interoperability engineer will define the FCR Schema such that it contains attribute and operation sets that exhibit a one-to-one correspondence with the attribute and operation sets of the System A ArmoredCombatVehicle_CCR_Schema and whose operations are behaviorally equivalent. In addition to the ArmoredCombatVehicle_CCR_Schema, this FEV will include schemas from all models of the real-world

entity having the same view as System A's *ArmoredCombatVehicle_CCR_Schema*. Although the component and federation models included with an FEV share the same view of the real-world entity, each may represent their defining attribute and operation sets differently.

Names used for the FCR Schema, as well as names and representations used in its attribute and operation sets, will be taken from the *Federation Ontology* as discussed in Section V.D.2.a(1). Similarly, the FE and FEV names will be taken from the Federation Ontology and will correspond with the names used for the FCR Schema. For our example, the new FE will be named *groundCombatVehicle*. The new FEV that has the same view of the real-world entity as the System A *ArmoredCombatVehicle_CCR_Schema* will be named *groundCombatVehicle_View1*. Lastly, the corresponding FCR and FCR Schema will be named *groundCombatVehicle_View1_FCR* and *GroundCombatVehicle_View1_FCR_Schema*, respectively. Figure A-2 illustrates this new FE with FEV containing both the FCR and CCR with their constituent Schemas, Syntax, and Semantics components. An association between the included FCR and CCR Schemas is also included to indicate the one-to-one correspondence between their attribute and operation sets.

B. ADDING COMPONENT CLASS REPRESENTATION (CCR) TO EXISTING FEDERATION ENTITY (FE)

If an FE exists in the FIOM for the real-world entity modeled by the CCR being registered, then the interoperability engineer must either find an existing FEV defined for the FE that has the same view of the real-world entity as the CCR, or must add such an FEV to the FE. The views (FEVs) of the FE are examined to determine the relationship between the Schema properties of the CCR being registered and those of the FCR defined for each of the FE's views. The correlation methodology described in Section VI.B can be used to assist the Interoperability Engineer in making this determination. This assistance is limited in the initial prototype OOMI IDE to helping find the FEV whose FCR Schema provides the closest match to the CCR Schema being registered; however, expansion of this assistance to identify the relationships between individual attributes and operations has been identified as an area for future research.

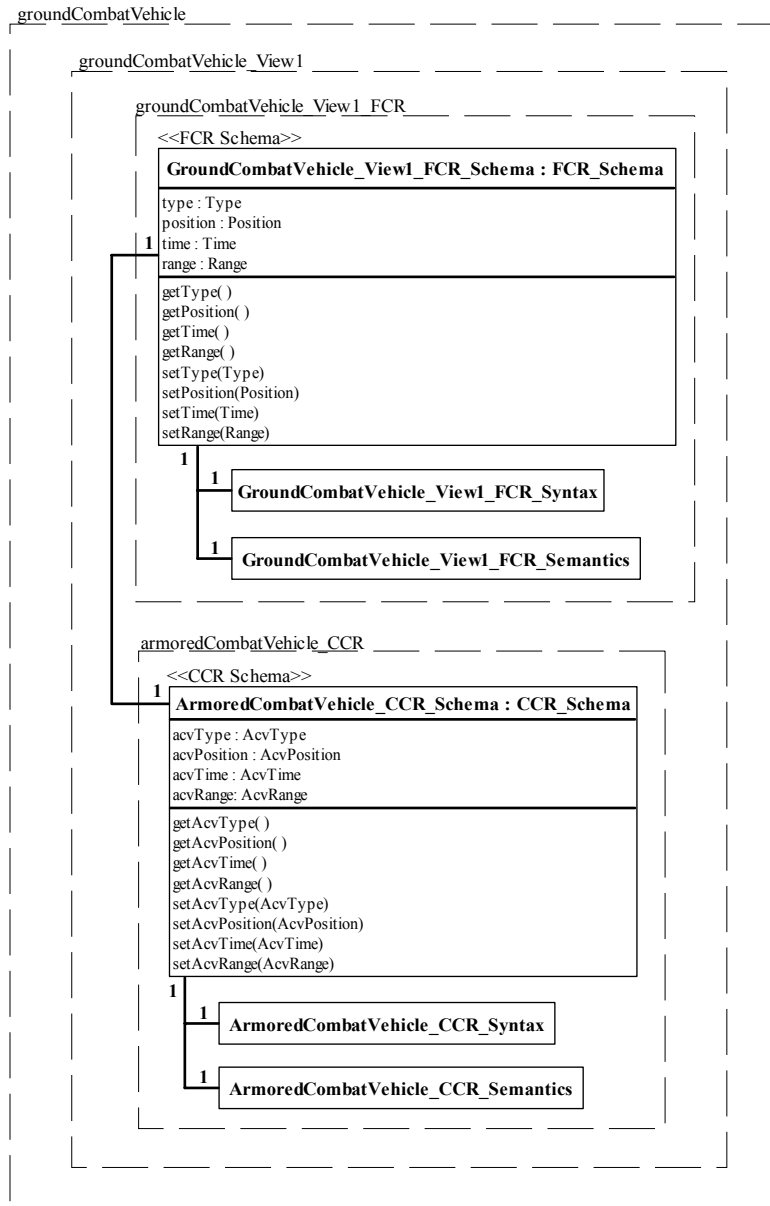


Figure A-2. New FE Defined With FEV Having Same Perspective of Real-World Entity as CCR Being Registered

1. Adding CCR to Existing Federation Entity View (FEV)

If there is an existing FEV defined for the FE whose FCR Schema attribute and operation sets exhibit a one-to-one correspondence with the Schema attribute and operation sets of the CCR being registered, then the CCR, with component CCR Schema, CCR Syntax, and CCR Semantics, is included with the FEV. Such would be the case if there exists a function $f: \text{CCR} \rightarrow \text{FCR}$ mapping the CCR Schema attribute and operation

sets ($CCR(A\mathcal{E}, \Omega\mathcal{E})$) to FCR Schema attribute and operation sets ($FCR(A\mathcal{E}, \Omega\mathcal{E})$) that is *one-to-one* and *onto* and whose operations are behaviorally equivalent. Automation of the process for determining behavioral equivalence has been identified as an area for further research.

Continuing the registration example begun in Appendix A Section A, suppose the attributes and operations for System B's `MechanizedCombatVehicle_CCR_Schema` corresponded to the attributes and operations of the `GroundCombatVehicle_View1_FCR_Schema` as depicted in Figure A-3 below. That is, there is a function $f_1: CCR \rightarrow FCR$ that can be defined mapping `MechanizedCombatVehicle_CCR_Schema` attribute and operation sets to `GroundCombatVehicle_View1_FCR_Schema` attribute and operation sets that is one-to-one and onto. In addition, suppose their operations are behaviorally equivalent. In this case System B's `mechanizedCombatVehicle_CCR`, with constituent components `MechanizedCombatVehicle_CCR_Schema`, `MechanizedCombatVehicle_CCR_Syntax`, and `MechanizedCombatVehicle_CCR_Semantics`, would be included with `groundCombatVehicle_View1` and associations established between the `MechanizedCombatVehicle_CCR_Schema` and the `GroundCombatVehicle_View1_FCR_Schema` as shown in Figure A-4.

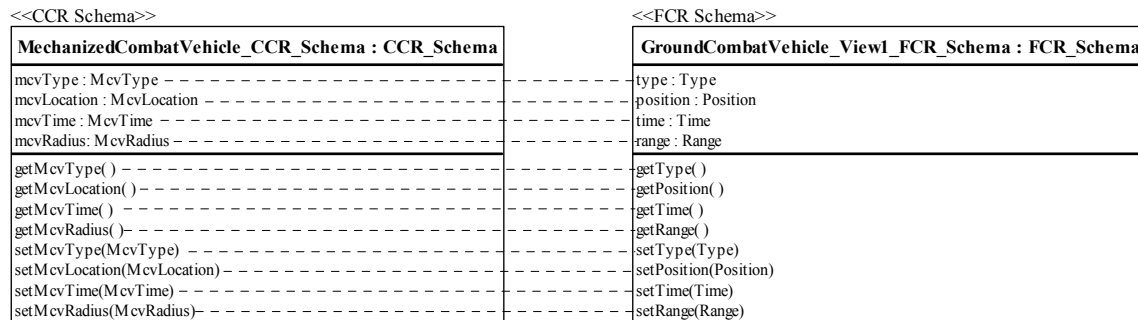


Figure A-3. One-To-One Correspondence Between CCR and FCR Schema Properties

2. Adding New View to FE

If the FE does not include a view whose FCR Schema attribute and operation sets have a one-to-one correspondence with the attribute and operation sets of the CCR Schema being registered and whose operation sets are behaviorally equivalent, then a new FEV with FCR Schema meeting these criteria must be defined for the FE. The new

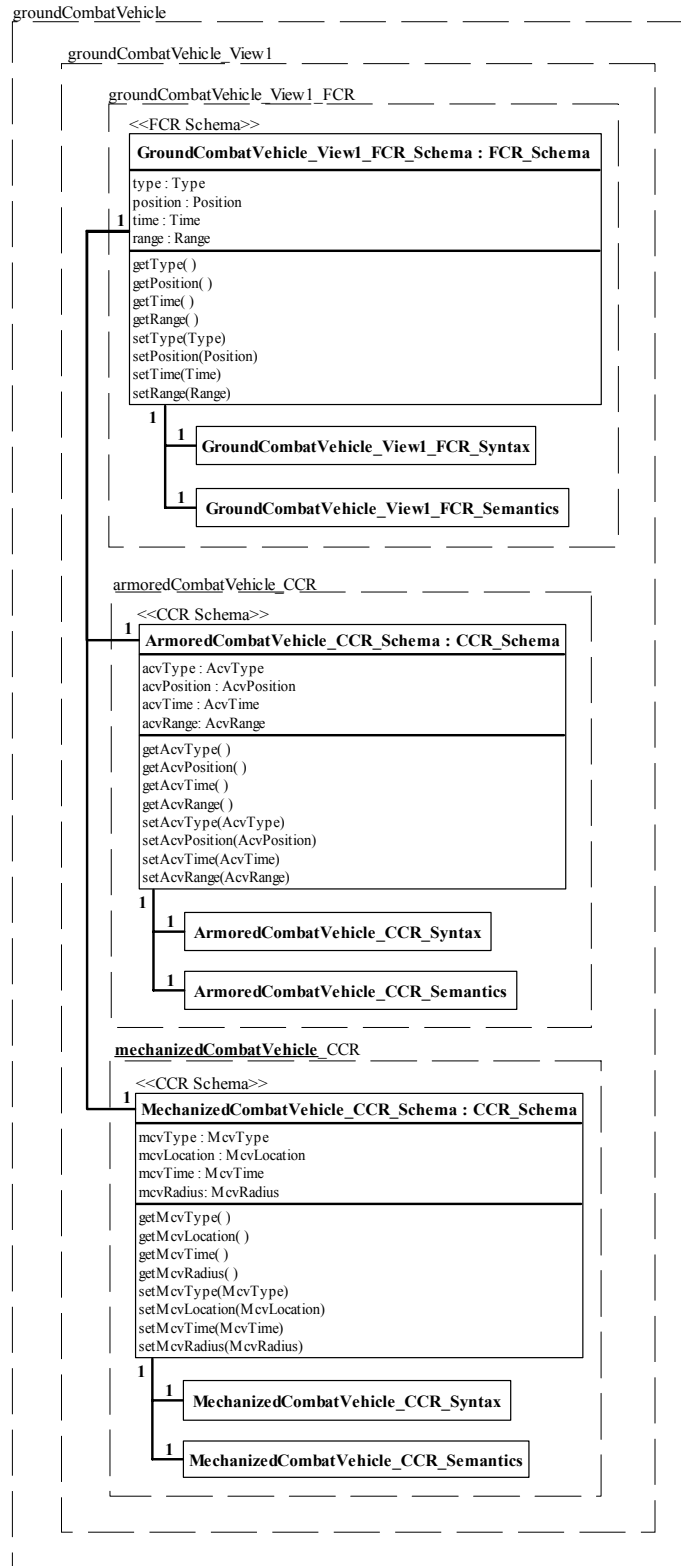


Figure A-4. CCR Added to Federation Entity View Exhibiting One-To-One Correspondence Between CCR and FCR Schema Properties

FEV will be derived from an existing view of the FE defined for the real-world entity modeled by the CCR being registered. The new FEV's defining FCR Schema will be constructed either through specialization or generalization of the existing view's FCR Schema. The choice of FCR Schema to specialize or generalize will be determined by the interoperability engineer, but the FCR Schema having the closest correspondence with the attribute and operation sets of the CCR Schema being registered should be selected. The correlation methodology described in Section VI.B can be used to assist the Interoperability Engineer in making this decision. Determination of whether the new FEV's FCR Schema will be defined by specialization or generalization of the selected FEV's FCR Schema will be dependent upon the relationship between CCR and FCR Schema attribute and operation sets. Assistance in establishing the relationships between individual attributes and operations has been identified as an area for future research.

a. CCR Schema Properties Subset of FCR Schema Properties

If, for the FEV selected, there is a function $f: \text{CCR} \rightarrow \text{FCR}$ mapping the CCR Schema attribute and operation sets ($\text{CCR}(\mathcal{A}\mathcal{E}, \mathcal{O}\mathcal{E})$) to the FEV's FCR Schema attribute and operation sets ($\text{FCR}(\mathcal{A}\mathcal{E}, \mathcal{O}\mathcal{E})$) that is *one-to-one* but not *onto*, then the interoperability engineer should define a new FEV with FCR Schema that generalizes the selected existing FEV's FCR Schema. Attribute and operation sets for the new FEV's FCR Schema should be created such that they exhibit a one-to-one correspondence with the attribute and operation sets of the CCR Schema being registered. In addition, in order to prevent difficulties associated with multiple inheritance, the selected FCR Schema should not already have a supertype in the FCR Schema Inheritance Hierarchy. If the selected FCR Schema does already extend another FCR Schema, another FEV should be selected as the basis for registering the CCR.

For example, in Figure A-5 the interoperability engineer selects ground-CombatVehicle_View1 as the view whose FCR Schema has the closest correspondence with the attribute and operation sets of the CCR Schema being registered, Armored-MilitaryVehicle_CCR_Schema. For these schemas, a *one-to-one* function $f_2: \text{CCR} \rightarrow \text{FCR}$ can be defined mapping ArmoredMilitaryVehicle_CCR_Schema attributes *designation*, *position*, and *time* and the related *get* and *set* operations for these

attributes to GroundCombatVehicle_View1_FCR_Schema attributes *type*, *position* and *time* and corresponding get and set operations as depicted. However, f_2 : CCR $\rightarrow$ FCR is not *onto* since GroundCombatVehicle_View1_FCR_Schema attribute *range* and related get and set operations do not have corresponding elements in the domain schema, ArmoredMilitaryVehicle_CCR_Schema. Therefore the interoperability engineer should define a new FEV with FCR Schema that generalizes GroundCombatVehicle_View1_FCR_Schema.

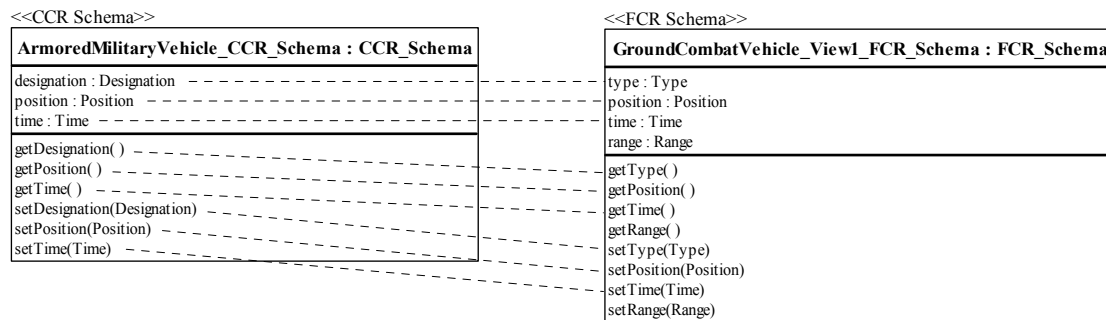


Figure A-5. CCR Schema Properties Subset of FCR Schema Properties

The new FEV's FCR Schema will contain attributes and operations from the selected existing FEV's FCR Schema that correspond to the attributes and operations contained in the CCR Schema being registered. This will provide a one-to-one correspondence between the attribute and operation sets of the CCR Schema being registered and the new FEV's FCR Schema. The selected existing FEV's FCR Schema will be modified such that it inherits the attributes and operations from the newly created FEV's FCR Schema and contains the remaining attributes and operations originally part of the existing FEV's FCR Schema that are not inherited from the new supertype FCR Schema. New FCR Syntax and Semantics components will be generated from the new supertype FCR Schema. FCR Syntax and Semantics components for the existing FCR are unaffected. The CCR, with component CCR Schema, CCR Syntax, and CCR Semantics, will then be included with the new FEV and an association established between the new FEV's FCR Schema and the CCR Schema being registered. Assistance will be provided by the Integrated Development Environment (IDE) in creating the new view with component FCR and FCR Schema that generalizes the selected existing FEV's

FCR Schema; however the interoperability engineer must identify the FCR Schema attributes and operations in the selected existing FEV that correspond to the CCR Schema attributes and operations.

From the example FCR Schema and CCR Schema correspondence shown in Figure A-5, a new FEV, `groundCombatVehicle_View2`, is defined from the view selected by the interoperability engineer, `groundCombatVehicle_View1`, and included with the `groundCombatVehicle` FE. As seen in Figure A-6, `GroundCombatVehicle_View2_FCR_Schema` generalizes `GroundCombatVehicle_View1_FCR_Schema` and contains attributes *type*, *position*, and *time* and their related get and set operations from `GroundCombatVehicle_View1_FCR_Schema`. These are the attributes and operations from `GroundCombatVehicle_View1_FCR_Schema` that correspond to the attributes and operations contained in `ArmoredMilitaryVehicle_CCR_Schema`. This provides the required one-to-one correspondence between the attribute and operation sets of the CCR Schema being registered and the corresponding FEV's FCR Schema.

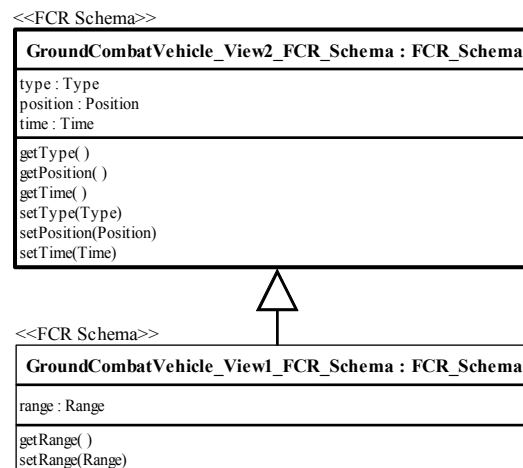


Figure A-6. Generalized FCR Schema Added to FEV

As shown in Figure A-7, `groundCombatVehicle_View2_FCR`, with constituent FCR Schema, Syntax, and Semantics components, is included with the new `groundCombatVehicle_View2` FEV. Subsequently, `armoredMilitaryVehicle_CCR`, with constituent CCR Schema, Syntax, and Semantics components, is then included with `groundCombatVehicle_View2` and associations established between the CCR Schema and FCR Schema for future reference by the translator. Syntax and semantics

components for the FCR and CCR as well as previously registered CCRs for ground-CombatVehicle_View1 have been omitted to enhance readability.

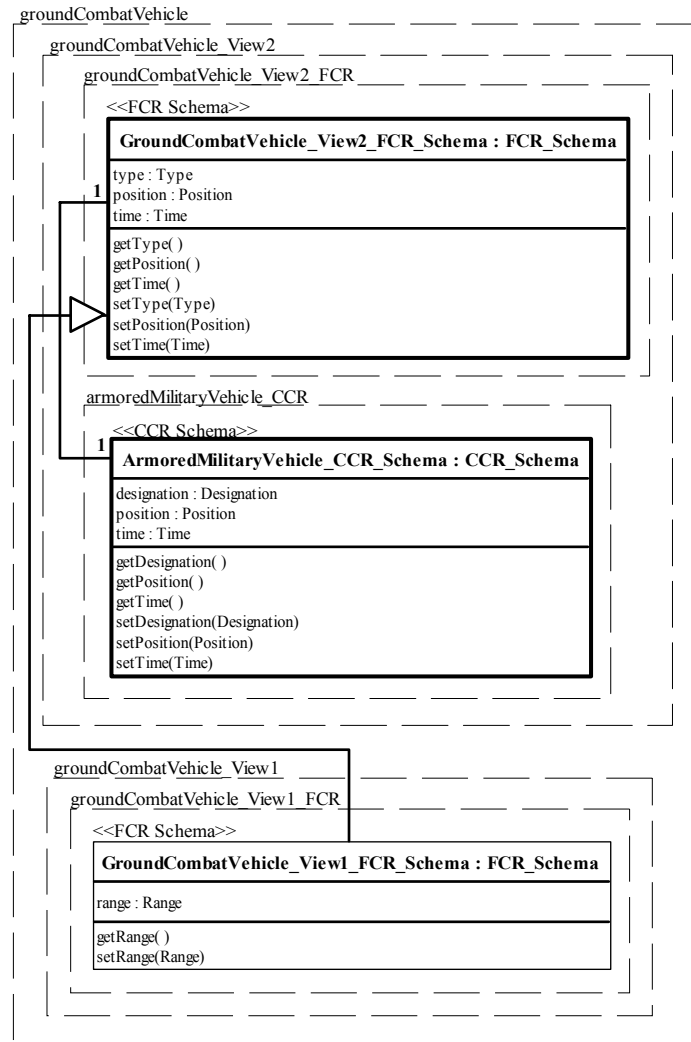


Figure A-7. FEV Containing Generalized FCR Schema Defined for FE

b. FCR Schema Properties Subset of CCR Schema Properties

If, for the FEV selected, there is a function $f: \text{FCR} \rightarrow \text{CCR}$ mapping the FEV's FCR Schema attribute and operation sets ($\text{FCR}(\mathcal{A}_\varepsilon, \Omega_\varepsilon)$) to CCR Schema attribute and operation sets ($\text{CCR}(\mathcal{A}_\varepsilon, \Omega_\varepsilon)$) that is one-to-one but not onto, then the interoperability engineer should define a new FEV whose FCR Schema specializes the selected existing FEV's FCR Schema. Attribute and operation sets for the new FEV's FCR

Schema should be created such that they exhibit a one-to-one correspondence with the attribute and operation sets of the CCR Schema being registered. For example, in Figure A-8 the interoperability engineer selects `groundCombatVehicle_View2` as the view whose FCR Schema has the closest correspondence with the attributes and operations of the next CCR Schema being registered, System D's `ArmoredFightingVehicle_CCR_Schema`. For these schemas, a one-to-one function $f_3: \text{FCR} \rightarrow \text{CCR}$ can be defined mapping `GroundCombatVehicle_View2_FCR_Schema` attributes *type*, *position*, and *time*, and related get and set operations to `ArmoredFightingVehicle_CCR_Schema` attributes *afvClassification*, *afvLocation*, and *afvObsTime* and its related get and set operations. However, $f_3: \text{FCR} \rightarrow \text{CCR}$ is not onto since `ArmoredFightingVehicle_CCR_Schema` attribute *afvStatus* and operations *getAfvStatus()* and *setAfvStatus(AfvStatus)* do not have corresponding elements in the domain schema, `GroundCombatVehicle_View2_FCR_Schema`. Therefore the interoperability engineer should define a new FEV with FCR Schema that extends `GroundCombatVehicle_View2_FCR_Schema` and contains attribute and operation sets ($\text{FCR}(\mathcal{A}\epsilon, \mathcal{O}\epsilon)$) that exhibit a one-to-one correspondence with `ArmoredFightingVehicle_CCR_Schema` attribute and operation sets and whose operations are behaviorally equivalent to the CCR Schema's operations.

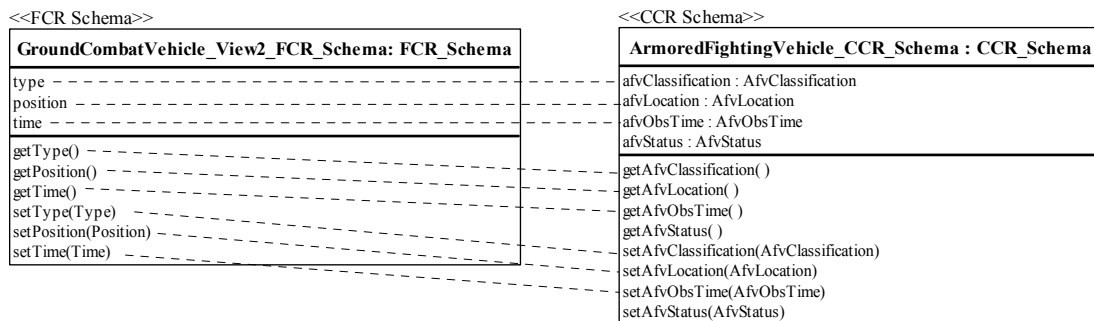


Figure A-8. FCR Schema Properties Subset of CCR Schema Properties

The new FEV's FCR Schema will inherit the attributes and operations defined for the existing FEV's FCR Schema being extended. The new FEV's FCR Schema will also include additional attributes and operations corresponding to the CCR Schema attributes and operations that are not represented in the existing FEV's FCR

Schema. This will provide a one-to-one correspondence between the attribute and operation sets of the new FEV's FCR Schema and the CCR Schema being registered. New FCR Syntax and FCR Semantics components that include syntactic and semantic information for the attributes and operations inherited from the existing FCR Schema will be generated from the new FCR Schema component. The CCR, with component CCR Schema, CCR Syntax, and CCR Semantics, will then be included with the new FEV and an association established between the new FEV's FCR Schema and the CCR Schema being registered. Assistance will be provided by the IDE in creating the new FEV with component FCR and FCR Schema that extends the selected existing FEV's FCR Schema; however definition of FCR Schema attributes and operations for the new FEV that correspond to the CCR Schema attributes and operations must be done by the interoperability engineer using information contained in the Federation Ontology.

Continuing our example from the FCR Schema and CCR Schema correspondence shown in Figure A-8, a new FEV, `groundCombatVehicle_View3`, is defined from the view selected by the interoperability engineer, `groundCombatVehicle_View2`, and included with the `groundCombatVehicle` FE. As seen in Figure A-9, `GroundCombatVehicle_View3_FCR_Schema` extends `GroundCombatVehicle_View2_FCR_Schema`. It includes additional properties, attribute *status* and operations *getStatus()* and *setStatus(Status)*, which correspond to `ArmoredFighting`

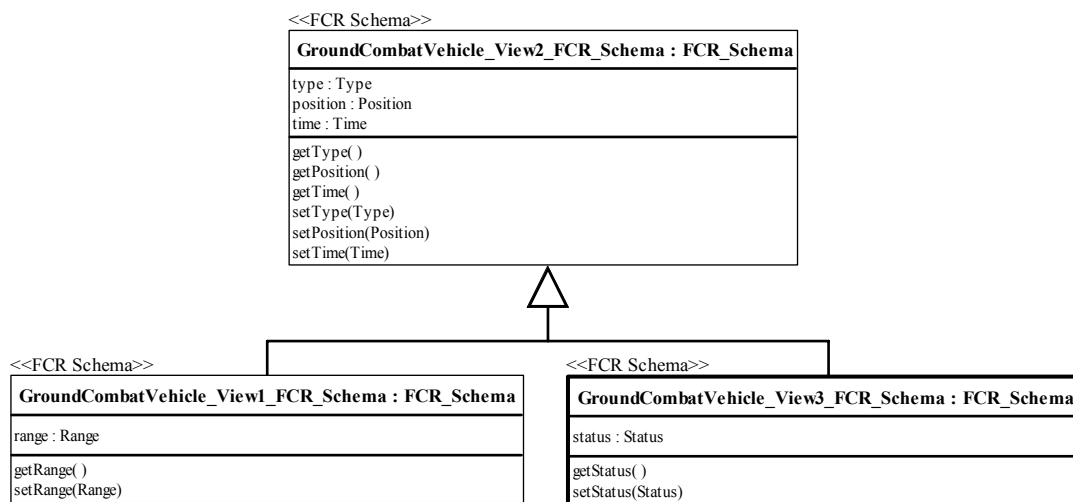


Figure A-9. Specialized FCR Schema Added to FEV

Vehicle_CCR_Schema attribute *afvStatus* and operations *getAfvStatus()* and *setAfvStatus(AfvStatus)*, respectively. This provides the required one-to-one correspondence between the attribute and operation sets of ArmoredFightingVehicle_CCR_Schema and GroundCombatVehicle_View3_FCR_Schema.

As shown in Figure A-10, groundCombatVehicle_View3_FCR, with constituent FCR Schema, Syntax, and Semantics, is included with the groundCombatVehicle_View3 FEV. Subsequently, armoredFightingVehicle_CCR, with constituent CCR Schema, Syntax, and Semantics components, is then included with groundCombatVehicle_View3 and associations established between the CCR Schema and FCR Schema for future reference by the translator. Syntax and semantics components for the FCRs and CCRs as well as CCRs for previously registered components have been omitted to enhance readability.

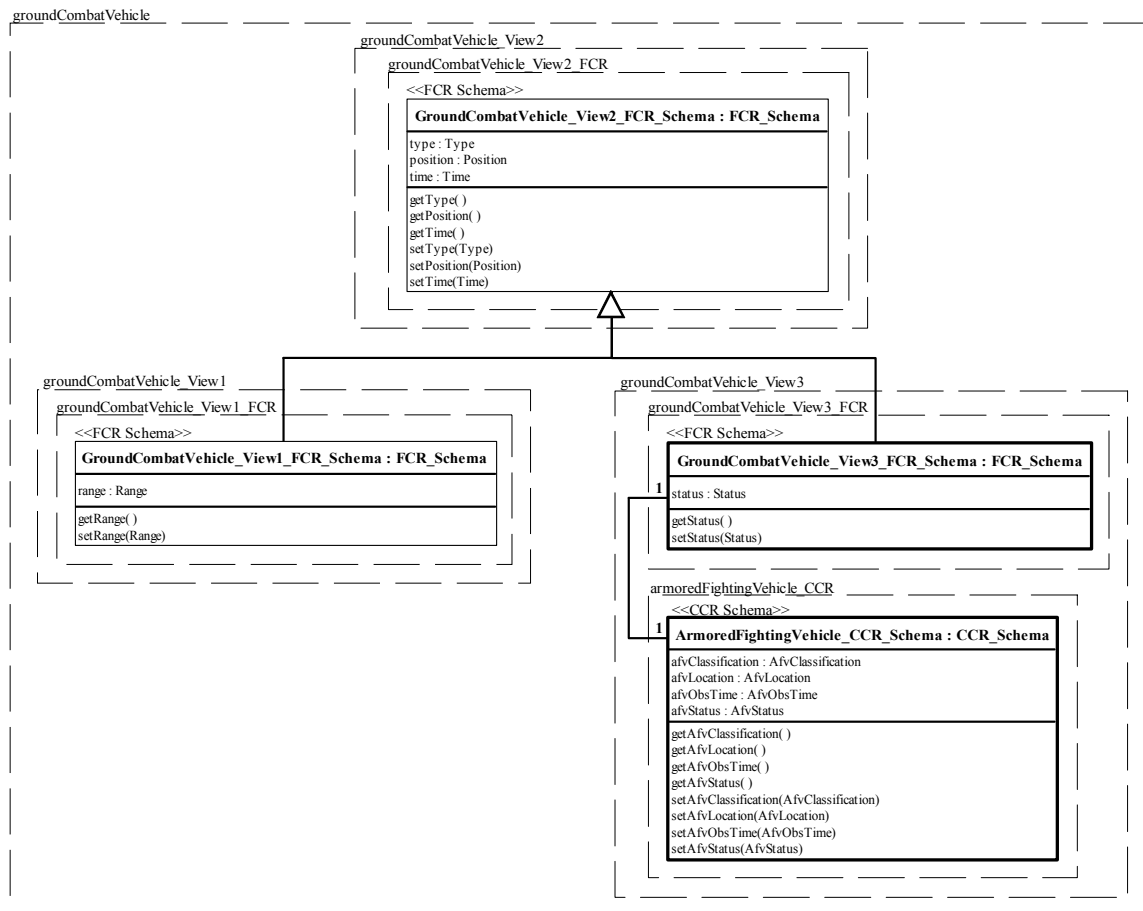


Figure A-10. FEV Containing Specialized FCR Schema Added to FE

c. CCR and FCR Schema Have Properties in Common, But No Subset Relation Exists.

Another possibility is for the CCR Schema and the selected FEV's FCR Schema to have corresponding attributes or operations, but that no function $f: \text{FCR} \rightarrow \text{CCR}$ or $g: \text{CCR} \rightarrow \text{FCR}$ mapping FCR Schema and CCR Schema attribute and operation sets can be defined. In this case we can define a function $f: \text{subset}(\text{CCR}) \rightarrow \text{subset}(\text{FCR})$ mapping a subset of the CCR Schema attributes and operations to a corresponding subset of the FCR Schema attributes and operations that is *one-to-one* and *onto*. For this situation the interoperability engineer should define a new FEV with FCR Schema that generalizes the selected existing FEV's FCR Schema, with the generalized FCR Schema containing attributes and operations from the existing FEV's FCR Schema that have corresponding attributes and operations in the CCR Schema being registered. In addition, a second new FEV should be defined with FCR Schema that specializes the first new FEV's FCR Schema and contains such additional attributes and operations required to provide a one-to-one correspondence with the attribute and operation sets of the CCR Schema being registered. Again, if the selected FEV's FCR Schema already extends another FCR Schema, a different FEV should be selected as the basis for registering the CCR in order to prevent problems associated with multiple inheritance.

To illustrate this case, I add an additional component system model of the ground combat vehicle real-world entity to those introduced in Figure IV-3. This model, from a component System E, names our real-world entity *Tank* and includes attributes *position*, *observationTime* and *radiusOfAction* and includes related get and set operations to characterize our ground combat vehicle from its perspective. In registering the CCR Schema for the System E Tank, the interoperability engineer selects groundCombatVehicle_View2 as the view whose FCR Schema has the closest correspondence to the attributes and operations of the Tank_CCR_Schema, as seen in Figure A-11. For these schemas, a *one-to-one*, *onto* function $f_4: \text{subset}(\text{CCR}) \rightarrow \text{subset}(\text{FCR})$ can be defined mapping Tank_CCR_Schema attributes *position* and *observationTime* and their related

get and set operations to GroundCombatVehicle_View2_FCR_Schema attributes *position* and *time* and related get and set operations as depicted.

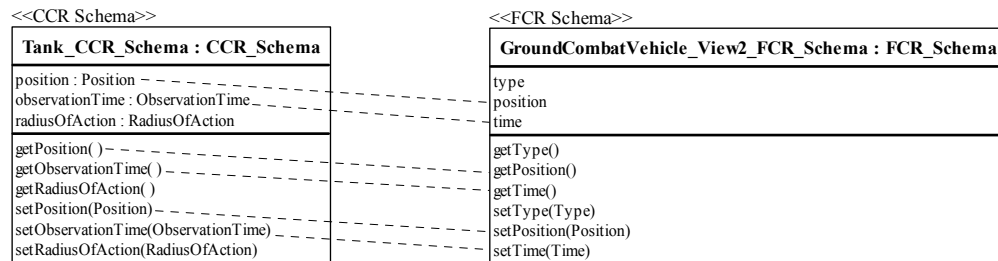


Figure A-11. CCR and FCR Schema Have Properties in Common, But No Subset Relation Exists

A new FEV whose FCR Schema generalizes the selected existing FEV's FCR Schema, containing the subset of the attributes and operations from the existing FEV's FCR Schema that corresponds to those of the CCR Schema being registered, is defined by the interoperability engineer. The selected existing FEV's FCR Schema will be modified such that it inherits the attributes and operations from the newly defined FEV's FCR Schema and contains additional attributes and operations originally part of the selected existing FEV's FCR Schema that are not inherited from the new FEV's FCR Schema. New FCR Syntax and FCR Semantics components will be generated from the new FEV's FCR Schema and included with the newly defined FEV. In addition, a second new FEV whose FCR Schema attribute and operation sets exhibit a one-to-one correspondence with the CCR Schema attribute and operation sets must be defined for the FE. This second new FEV's FCR Schema will extend the first new FEV's generalized FCR Schema. The second new FEV's specialized FCR Schema will inherit the attributes and operations defined for the first new FEV's generalized FCR Schema. The second new FEV's specialized FCR Schema will also include additional attributes and operations required to provide a one-to-one correspondence with the attribute and operation sets of the CCR Schema being registered.

New FCR Syntax and FCR Semantics components will be generated from this second new FEV's specialized FCR Schema and included with the FEV. The CCR, with component CCR Schema, CCR Syntax, and CCR Semantics, will then be included

with the second new FEV and an association established between the second new FEV's FCR Schema and the CCR Schema being registered. Assistance will be provided by the IDE in defining the new FEV with component FCR whose FCR Schema generalizes the selected existing FEV's FCR Schema. Also, assistance will be provided by the IDE in defining the second new FEV with component FCR whose FCR Schema specializes the first new FEV's FCR Schema. However, the interoperability engineer is responsible for identifying the FCR Schema attributes and operations in the selected existing FEV that correspond to the CCR Schema attributes and operations and for defining FCR Schema attributes and operations for the second new FEV's specialized FCR Schema that correspond to the CCR Schema attributes and operations that do not have a corresponding attribute or operation in the first new FEV's FCR Schema.

From the example FCR Schema and CCR Schema correspondence shown in Figure A-11, a new FEV, `groundCombatVehicle_View4`, is defined from the FEV selected by the interoperability engineer, `groundCombatVehicle_View2`, and included with the `groundCombatVehicle` FE. As seen in Figure A-12, `GroundCombatVehicle_View4_FCR_Schema` generalizes `GroundCombatVehicle_View2_FCR_Schema` and contains attributes *position* and *time* with related get and set operations from the `GroundCombatVehicle_View2_FCR_Schema`. These are the attributes and operations

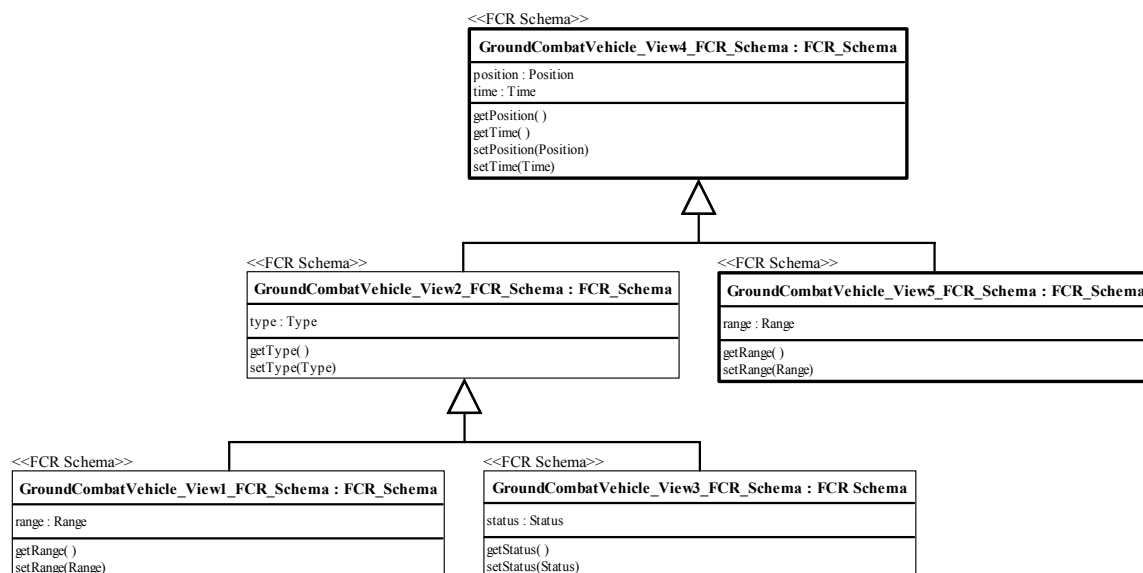


Figure A-12. Sibling to Existing FCR Schema Added to FEV

from GroundCombatVehicle_View2_FCR_Schema that have a corresponding attribute or operation in Tank_CCR_Schema. In addition, GroundCombatVehicle_View5_FCR_Schema extends GroundCombatVehicle_View4_FCR_Schema. It also includes additional properties, attribute *range* and operations *getRange()* and *setRange(Range)*, which correspond to Tank_CCR_Schema attribute *radiusOfAction* and operations *getRadiusOfAction()* and *setRadiusOfAction(RadiusOfAction)*, respectively. This provides the required one-to-one correspondence between the attributes and operations of the CCR Schema being registered, Tank_CCR_Schema, and the newly added FEV's FCR Schema, GroundCombatVehicle_View5_FCR_Schema.

As seen in Figure A-13, groundCombatVehicle_View4_FCR and groundCombatVehicle_View5_FCR, with constituent FCR Schema, FCR Syntax, and FCR Semantics components are included with groundCombatVehicle_View4 and groundCombatVehicle_View5, respectively. Subsequently, System E Tank_CCR, with

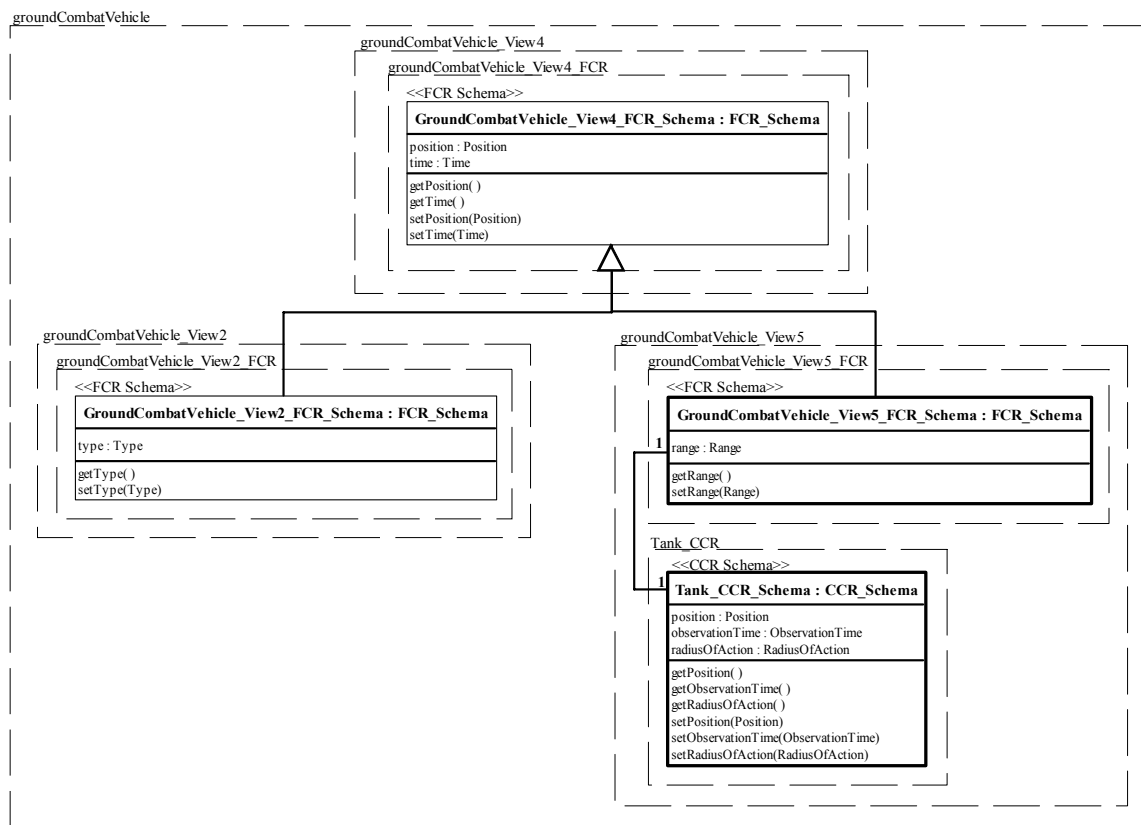


Figure A-13. FEV With FCR Schema Sibling to Existing FCR Schema Added to FE

constituent CCR Schema, CCR Syntax, and CCR Semantics components, is then included with groundCombatVehicle_View5 and associations established between the CCR Schema and FCR Schema for future reference by the translator. Again, syntax and semantics components for both the FCR and CCR have been omitted to enhance readability. Additionally, all FEVs whose FCR Schemas are descendants of Ground-CombatVehicle_View2_FCR_Schema have been omitted from the figure.

d. No Correspondence Between CCR and FCR Schemas

A final possible relationship between the CCR being registered and FIOM FEs is that an FE exists for the real-world entity modeled by the CCR being registered, but no correspondence can be found in the FE between the CCR Schema's attribute and operation sets and those of an existing FEV's FCR Schema. In this case the interoperability engineer must define a new FEV for the FE whose defining FCR Schema attribute and operation sets provide a one-to-one correspondence with the CCR Schema attribute and operation sets. While the newly defined FEV's FCR Schema has a totally different perspective of the real-world entity being modeled than do the other FCR Schemas for the FE (as determined by the lack of correspondence between their attribute and operation sets), it does model the same real-world entity captured by the other FEVs. Therefore, the relationship between this new FEV's FCR Schema and existing FCR Schemas for other FEVs in the FE should be reflected in the FE's FCR Schema Inheritance Hierarchy for future reference by the translator (discussed in Chapter VII). This is accomplished by including a new FEV whose FCR Schema will serve as the new root of the existing FCR Schema Inheritance Hierarchy for the FE. This new root FCR Schema will then be used to relate the existing FEVs to the new FEV whose FCR Schema corresponds to the schema of the CCR being registered.

This new FCR Schema will generalize the existing FCR Schema Inheritance Hierarchy root FCR Schema and will serve as the new root for the inheritance hierarchy. The new root FCR Schema will not contain any attributes or operations. In addition, a second new FEV should be defined whose FCR Schema specializes the first new FEV's FCR Schema and whose FCR Schema attribute and operation sets provide a one-to-one correspondence with the CCR Schema attribute and operation sets of the CCR being registered. Although it is considered unlikely in practice that two models of the

same real-world entity would not have any properties in common (otherwise they would not be able to interoperate), it is feasible that this situation could occur as an interim step during bottom-up construction of the FCR Schema Inheritance Hierarchy. In this case, the process described above is necessary in order to preserve connectivity between FCR Schema nodes comprising the FCR Schema Inheritance Hierarchy for later use by the translator.

To illustrate the case described above, I add yet another component system model of the ground combat vehicle real-world entity to those introduced in Figure IV-3 to my example. Instead of the command-and-control related systems introduced earlier, this last system, System F, is a logistics management system and requires information related to the provisioning of the modeled ground combat vehicle. As seen in Figure A-14, System F's model of the real-world entity is named *MainBattleTank* and includes attributes *fuelType* and *ammoType* for tracking the vehicle's provisions. The defining *MainBattleTank_CCR_Schema* also includes operations for *getting* and *setting* the provision type values.

Using the correlation software discussed in Chapter VI above to search the FIOM for federation entities related to the component system model being registered, the interoperability engineer discovers that the existing *groundCombatVehicle* FE provides a model of the same real-world entity as the *MainBattleTank_CCR_Schema* being registered. However, as depicted in Figure A-14 it is discovered that none of the views currently defined for the FE contain an FCR Schema that has any attributes or operations in common with the *MainBattleTank_CCR_Schema*'s attributes or operations.

In this case, the interoperability engineer defines a new FEV whose FCR Schema generalizes the existing root FCR Schema in the FE's FCR Schema Inheritance Hierarchy. This FCR Schema will function as the new root FCR Schema for the inheritance hierarchy and will contain no attributes or operations. The previously existing root FCR Schema will extend this newly created FCR Schema and will otherwise be unchanged. In addition, a second new FEV with FCR Schema whose attribute and operation sets exhibit a one-to-one correspondence with the CCR Schema attribute and operation sets must be defined for the FE. This second new FEV's FCR

Schema will extend the new root FCR Schema. The second new FEV's specialized FCR Schema will inherit the attributes and operations defined for the new root FCR Schema. The second new FEV's specialized FCR Schema will also include additional attributes and operations required to provide a one-to-one correspondence with the attribute and operation sets of the CCR Schema from the CCR being registered.

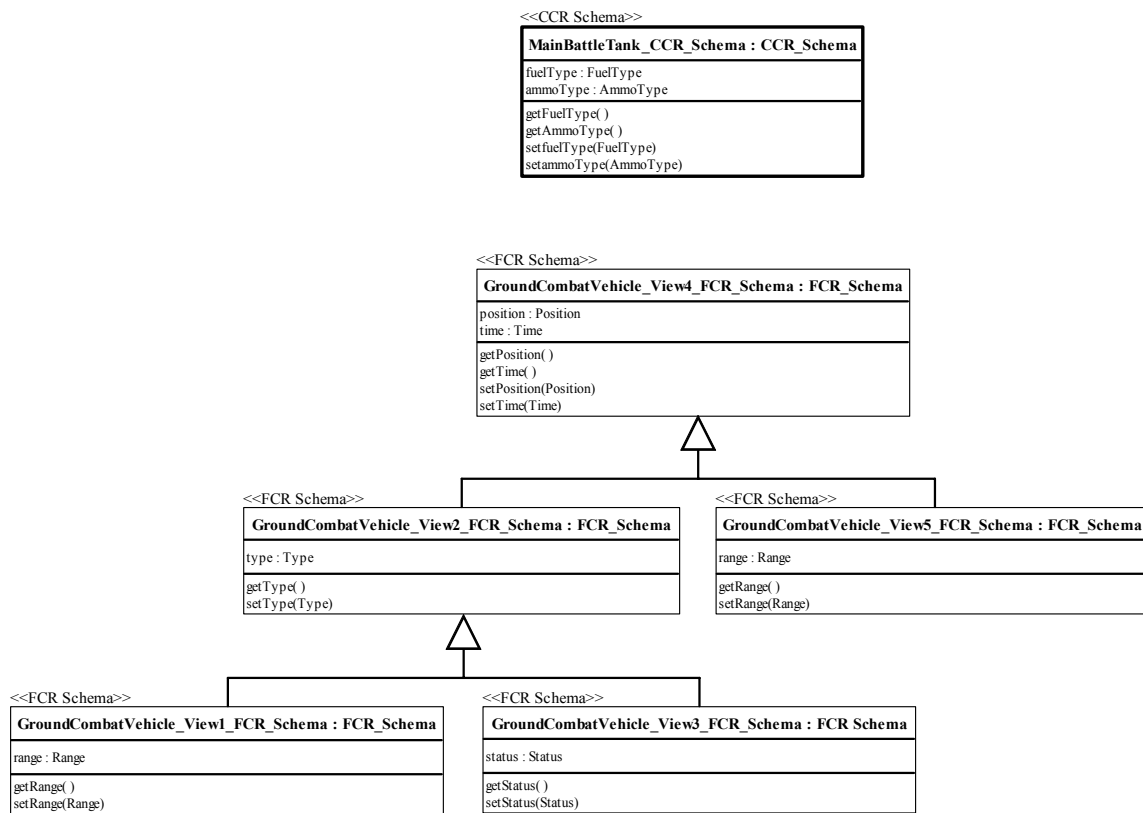


Figure A-14. No Correspondence Between CCR Schema Being Registered and Existing FCR Schemas in FE

New FCR Syntax and FCR Semantics components will be generated from this second new specialized FCR Schema and included with the FEV. The CCR, with component CCR Schema, CCR Syntax, and CCR Semantics, will then be included with the second new FEV and an association established between this FEV's FCR Schema and the CCR Schema. Assistance will be provided by the IDE in defining the new FEV whose component FCR Schema serves as the new FCR Schema Inheritance Hierarchy root and generalizes the previously existing root FCR Schema. Also, assistance will be

provided by the IDE in defining the second new FEV whose component FCR Schema specializes the new root FCR Schema. However, definition of the additional FCR Schema attributes and operations for the second new specialized FCR Schema that correspond to the CCR Schema attribute and operation sets must be done by the interoperability engineer.

For the example FCR Schema and CCR Schema correspondence shown in Figure A-14, a new FEV, `groundCombatVehicle_View6`, is defined from the view containing the previous existing root FCR Schema, `groundCombatVehicle_View4`, and included with the `groundCombatVehicle` FE. As seen in Figure A-15, `GroundCombatVehicle_View6_FCR_Schema` generalizes `GroundCombatVehicle_View4_FCR_Schema` and contains no attributes or operations. As a result, `GroundCombatVehicle_View6_FCR_Schema` becomes the new root for the FCR Schema Inheritance Hierarchy, leaving the previous root, `GroundCombatVehicle_View4_FCR_Schema`, otherwise unchanged. In addition, `GroundCombatVehicle_View7_FCR_Schema` extends `GroundCombatVehicle_View6_FCR_Schema` and includes attributes *fuelType* and *ammoType* and their related get and set operations corresponding to *MainBattle*

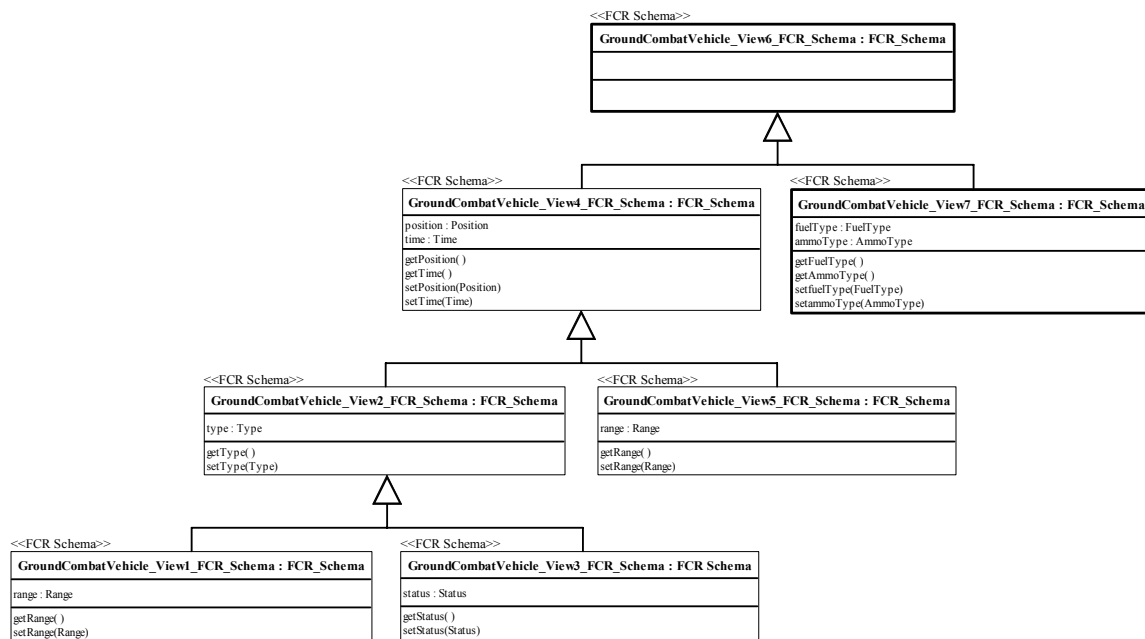


Figure A-15. New Root FCR Schema With Child FCR Schema Corresponding to CCR Schema Being Registered Included with FEV

Tank_CCR_Schema attributes *fuelType* and *ammoType* and their related get and set operations, respectively. This provides the required one-to-one correspondence between the attribute and operation sets of the CCR Schema being registered, MainBattleTank_CCR_Schema, and the newly defined FEV's FCR Schema, GroundCombatVehicle_View7_FCR_Schema.

As seen in Figure A-16, groundCombatVehicle_View6_FCR and groundCombatVehicle_View7_FCR, with constituent FCR Schema, FCR Syntax, and FCR Semantics components are included with groundCombatVehicle_View6 and groundCombatVehicle_View7, respectively. Subsequently, mainBattleTank CCR, with

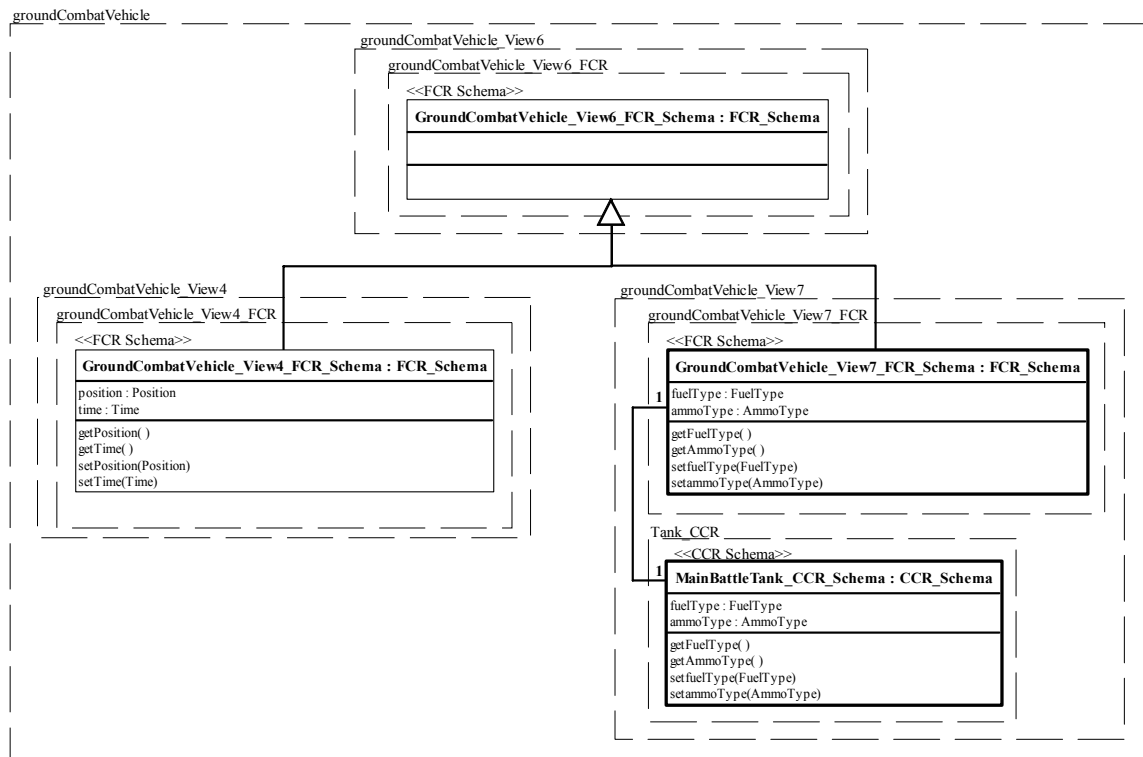


Figure A-16. FEV With FCR Schema Sibling to Previous Existing Root FCR Schema Added to FE

constituent CCR Schema, CCR Syntax, and CCR Semantics components, is then included with groundCombatVehicle_View7 and associations established between the CCR Schema and FCR Schema for future use by the translator. Syntax and semantics

components for the FCRs and CCRs as well as all FEVs whose FCR Schemas are descendants of GroundCombatVehicle_View4_FCR_Schema have been omitted to enhance readability.

C. SUMMARY

Appendix A details the modifications to the Federation Interoperability Object Model (FIOM) required in order to provide the required correspondence between component (CCR) and federation (FCR) class representation schemas during CCR registration. Procedures are presented for cases where 1) a new FE with Federation Entity View (FEV) must be defined for the FIOM to enable CCR registration or 2) an existing FE is used for CCR registration. If an existing FE is used, then either the CCR will be included with one of the FE's existing FEVs or with a new FEV defined by generalization or specialization of an existing FEV's FCR Schema. The OOMI IDE provides computer aid to the FIOM modification process during CCR registration as indicated in Appendix A Sections A and B.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES.

- [ABK00] Anderson, R., and others, *Professional XML*, Wrox Press Ltd., Birmingham, UK, 2000.
- [Ant94] Anton, H., *Elementary Linear Algebra*, John Wiley & Sons, Inc., New York, NY, 1994.
- [Ber00] Berg, C., *Advanced Java 2 Development for Enterprise Applications*, 2d ed., Sun Microsystems Press, Prentice-Hall, Inc., Upper Saddle River, New Jersey, 2000.
- [BFH+95] Benkley, S., and others, *Data Element Tool-Based Analysis (DELTA)*, MITRE Corporation report MTR95B0000147, December 1995.
- [BM01] Biron, P., Malhotra, A. (editors), “XML Schema Part 2: Datatypes”, [<http://www.w3.org/TR/xmlschema-2/>], 2 May 2001.
- [BOD01] Birbeck, M., and others, *Professional XML, 2nd Edition*, Wrox Press Ltd., Birmingham, UK, 2001.
- [BLN86] Batinin, C., Lenzerini, M., Navathe, S., “A Comparative Analysis of Methodologies for Database Schema Integration”, *ACM Computing Surveys*, Volume 18, No. 4, pp. 323-364, 1986.
- [BRJ99] Booch, G., Rumbaugh, J., Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley Longman, Inc., Redding, MA, 1998.
- [CHR97] Clifton, C., Housman, E., Rosenthal, A., “Experience with a Combined Approach to Attribute-Matching Across Heterogeneous Databases”, IFIP 1997, Chapman & Hall.
- [CORBA01] *The Common Object Request Broker: Architecture and Specification*, Revision 2.6, Object Management Group, Inc., December 2001.
- [CY01] Christie, B., Young, P., *Integrated Development Environment (IDE) for Construction of a Federation Interoperability Object Model (FIOM)*, Master’s thesis, Naval Postgraduate School, Sep 2001.
- [DCOM96] “DCOM Technical Overview”, [http://www.msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp], November 14, 2001.

- [DDA91] Department of Defense (DoD) Directive 8320.1, *DoD Data Administration*, [<http://web7.whs.osd.mil/pdf/d83201p.pdf>], Sep 26, 1991.
- [DII01] “DII COE Data Emporium.”
[<http://diides.ncr.disa.mil/xmlreg/user/index.cfm>].
- [EGI00] SeeBeyond White Paper 00403.102, “e*Gate Integrator, The eBusiness Integration Platform”, Software Technologies Corporation, November 2000.
- [EIGI00] SeeBeyond White Paper “e*Index Global Identifier™ Enabling a Single Customer View”, SeeBeyond Technology Corporation, November 2000.
- [FDM01] “Functional Description of the Mission Space.”
[<http://fdms.msiac.dmsomil/>].
- [Fei99] Feit, S., *TCP/IP*, McGraw-Hill, New York, NY, 1999.
- [GL99] Guo, J., Luqi, “Object Modeling to Re-engineer Legacy Systems”, *The Eleventh International Conference On Software Engineering And Knowledge Engineering (SEKE 99)*, June 17 to 19, 1999, pp. 346-353.
- [GNM96] Goguen, J., Nguyen, D., Meseguer, J., Luqi, Zhang, D., Berzins, V., “Software Component Search”, *Journal of Systems Integration*, Volume 6, No. 2, 1996, pp. 93-134.
- [Gra87] Grant, J., *Logical Introduction to Databases*, Harcourt Brace Jovanovich, Inc., San Diego, CA, 1987.
- [GW88] SRI International Report SRI-CSL-88-9, “Introducing OBJ3”, Goguen, J., and Winkler, T., August 1988.
- [Her97] Herman, J.S., *Improving Syntactic Matching for Multi-level Filtering*, Master’s thesis, Naval Postgraduate School, 1997.
- [HL96] Holowczak, R., Li, W., “A Survey on Attribute Correspondence and Heterogeneity Metadata Representation”,
[<http://www.computer.org/conferences/meta96/li/paper.html>], 1996.
- [HLA02] “High Level Architecture”,
[<http://www.dmsomil/public/transition/hla/>].
- [HM99] Hammer, J., McLeod, D., “Resolution of Representational Diversity in Multidatabase Systems”, *Management of Heterogeneous and Autonomous Database Systems*, Morgan Kaufman, 1999.

- [KA95] Khoshafian, S., Abnous, R., *Object Orientation*, John Wiley and Sons, Inc., New York, NY, 1995.
- [Kay00] Kay, M., *XSLT Programmer's Reference*, Wrox Press Ltd., Birmingham, UK, 2000.
- [Kir97] Kirtland, M., "The COM+ Programming Model Makes it Easy to Write Components in Any Language", *Microsoft Systems Journal*, December 1997.
- [KM98] Kahng, J., McLeod D., "Dynamic Classificational Ontologies: Mediation of Information Sharing in Cooperative Federated Database Systems", *Cooperative Information Systems, Trends and Directions*, Academic Press, 1998.
- [Koh87] Kohonen, T., "Adaptive Associative and Self-Organizing Functions in Neural Computing", *Applied Optics*, Vol. 26, 1987, pp. 4910–4918.
- [Kol00] Kolb, R., "An Introduction to COM, DCOM and COM+", [http://www.stolles.net/fsu-swt/LV/CompSem2000/works/COM-paper_html/], October 2000.
- [KS91] Kim, W., Seo, J., "Classifying Schematic and Data Heterogeneity in Multidatabase Systems", *IEEE Computer*, Vol. 24, No. 12, pp. 12-18, December, 1991.
- [KWD99] Kuhl, F., Weatherly, R., Dahmann, J., *Creating Computer Simulation Systems, An Introduction to the High Level Architecture*, Prentice Hall PTR, Upper Saddle River, NJ, 1999.
- [Law01] Lawler, G., "Federation Session Management Protocol (FSMP)", SPAWARSYSCEN San Diego Research Fellowship Proposal, October, 2001.
- [LB88] Luqi, Berzins, V., "Rapidly Prototyping Real-Time Systems", *IEEE Software*, pp. 25-36, September 1988.
- [LBY88] Luqi, Berzins, V., Yeh, R., "A Prototyping Language for Real-Time Software", *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, October 1988, pp. 1409-1423.
- [LC94] Li, W., and Clifton, C., "Semantic Integration in Heterogeneous Databases Using Neural Networks", *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994, pp. 1-12.

- [LC00] Li, W., and Clifton, C., "SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks", *Data & Knowledge Engineering*, Vol. 33 (2000), pp. 49-84.
- [Lee02] Lee, S., *Class Translator for the Federation Interoperability Object Model (FIOM)*, Master's thesis, Naval Postgraduate School, Mar 2002.
- [LISI98] *Levels of Information Systems Interoperability (LISI)*, C4ISR Architecture Working Group, 30 March 1998.
- [LW94] Liskov, B., Wing, J., "A Behavioral Notion of Subtyping," *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 6, November 1994, pp. 1811-1841.
- [Ngu95] Nguyen, D., *An Architectural Model for Software Component Search*, Ph.D. dissertation, Naval Postgraduate School, 1995.
- [OMG01] "About the Object Management Group."
[<http://www.omg.org/gettingstarted/gettingstartedindex.htm>].
- [Pit97] Pitoura, E., "Providing Database Interoperability through Object-Oriented Language Constructs", *Journal of Systems Integration*, Volume 7, No. 2, August 1997, pp. 99-126.
- [Pop98] Pope, A., *The CORBA Reference Guide*, Addison-Wesley Longman, Inc., Redding, MA, 1998.
- [Pre01] Pressman, R., *Software Engineering A Practitioner's Approach*, McGraw-Hill, Boston, MA, 2001.
- [Pri91] Prieto-Diaz, R., "Implementing Faceted Classification for Software Reuse", *Communications of the ACM*, Volume 34, No. 5, May 1991.
- [Pug01] Pugh, R., *Methods For Determining Object Correspondence During System Integration*, Master's thesis, Naval Postgraduate School, Jun 2001.
- [Rei01] Reinhold, M., *The Java™ Architecture for XML Binding (JAXB)*, Working-Draft Specification, Sun Microsystems, Inc.,
[ftp://ftp.java.sun.com/pub/xml/987dfjaxb10ea3ds/jaxb-0_21-wd-spec.pdf], 30 May 2001.
- [Ros98] Rosenberger, J., *Teach Yourself CORBA in 14 Days*, Sams Publishing, Indianapolis, Indiana, 1998.

- [RRH00] Ralston, A., Reilly, E., and Hemmendinger, D., *Encyclopedia of Computer Science*, Fourth Edition, Nature Publishing Group, London, UK, 2000.
- [RRS96] Renner, S., Rosenthal, A., Scarana, J., “Data Interoperability: Standardization or Mediation”, *IEEE Metadata Workshop*, Silver Spring, MD, April 1996.
- [SG96] Shaw, M., Garlan, D., *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.
- [She02] Shedd, S., *Semantic and Syntactic Object Correlation in the Object-Oriented Method for Interoperability*, Master’s thesis, Naval Postgraduate School, Sep 2002.
- [Smi01] Smith, S., “eGate Technical Overview”, Presentation to Defense Manpower Data Center, June 2001.
- [Sta00] Stallings, W., *Data and Computer Communications, Sixth Edition*, Prentice-Hall, Inc., Upper Saddle River, NJ, 2000.
- [Ste91] Steigerwald, R. A., *Reusable Software Component Retrieval via Normalized Algebraic Specifications*, Ph.D. dissertation, Naval Postgraduate School, 1991.
- [SV02] Schmidt, D., and Vinoski, S., “Object Interconnections: CORBA and XML, Part 1: Versioning”, *C/C++ Users Journal*, [<http://www.cuj.com/experts/1905/vinoski.htm>]
- [TWS02] “Tomahawk Weapon System, Afloat Planning System, and Theater Mission Planning Center”, [<http://www.fas.org/man/dod-101/sys/ship/weaps/tmpc-aps.htm>].
- [WCS00] Walsh, A., Couch, J., Steinberg, D., *Java 2 Bible*, IDG Books Worldwide, Inc., Foster City, CA, 2000.
- [Wie93] Wiederhold, G., “Intelligent Integration of Information”, *ACM-SIGMOD 93*, Washington, DC, May 1993, pp. 434-437.
- [WK94] Williams, S., Kindel, C., “The Component Object Model: A Technical Overview”, [http://msdn.microsoft.com/library/en-us/dncomg/html/msdn_comppr.asp].
- [WO00] Wing, J.M., “Respectful Type Converters”, *IEEE Transactions on Software Engineering*, Volume 26, Number 7, July 2000, pp. 579-593.

- [YBG02] Young, P., Berzins, V. Ge, J., Luqi, “Using an Object Oriented Model for Resolving Representational Differences between Heterogeneous Systems”, *The 17th ACM Symposium on Applied Computing*, Madrid, Spain, March 10 – 14, 2002.
- [ZW93] Zaremski, A.M., Wing, J.W., “Signature Matching: A Key to Reuse”, *SIGSOFT '93*, ACM 0-8971-625-5/93/0012, pp. 182-190.
- [ZW95] Zaremski, A.M., Wing, J.W., “Specification Matching of Software Components”, *SIGSOFT '95*, ACM 0-89791-716-2/95/0010, pp.6-17.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center	2
	8725 John J. Kingman Road, Suite 0944	
	Ft. Belvoir, VA 22060-6218	
2.	Dudley Knox Library	2
	Naval Postgraduate School	
	411 Dyer Road	
	Monterey, CA 93943-5101	
3.	Professor Luqi	1
	Software Engineering Automation Center	
	Naval Postgraduate School	
	Monterey, CA 93943	
4.	Professor V. Berzins.....	1
	Software Engineering Automation Center	
	Naval Postgraduate School	
	Monterey, CA 93943	
5.	Professor G. Jun	1
	Software Engineering Automation Center	
	Naval Postgraduate School	
	Monterey, CA 93943	
6.	Professor W. Kemple	1
	Command, Control, and Communications	
	Naval Postgraduate School	
	Monterey, CA 93943	
7.	Mr. R. Riehle.....	1
	Software Engineering Automation Center	
	Naval Postgraduate School	
	Monterey, CA 93943	
8.	Dr. Edmund Freeman	1
	Vice President and Manager, Integrated Systems Technology Operation	
	Technology Research Group	
	Science Applications International Corporation	
	1213 Jefferson Davis Highway, Suite 1500, Arlington, VA 22202-4304	

9.	CAPT Paul Young.....	10
	305 Morning Dove Way	
	Arnold, MD 21012	