MODEL-BASED APPROACH
TO
THE FEDERATION OBJECT MODEL INDEPENDENCE PROBLEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MEHMET FATİH ULUAT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2007

Approval of the thesis:

MODEL-BASED APPROACH TO THE FEDERATION OBJECT MODEL INDEPENDENCE PROBLEM

submitted by **MEHMET FATİH ULUAT** in partial fulfillment of the requirements for the degree of **Master of Sciences in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Volkan Atalay _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Halit Oğuztüzün _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Assoc. Prof. Dr. Ferda Nur Alpaslan _____
Computer Engineering Dept., METU

Assoc. Prof. Dr. Halit Oğuztüzün _____
Computer Engineering Dept., METU

Assoc. Prof. Dr. Veysi İşler _____
Computer Engineering Dept., METU

Dr. Ayşenur Birtürk _____
Computer Engineering Dept., METU

Sami Duman _____
Manager, ASELSAN

Date: 21/08/2007

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Mehmet Fatih Uluat

Signature :

ABSTRACT

MODEL-BASED APPROACH TO THE FEDERATION OBJECT MODEL INDEPENDENCE PROBLEM

Uluat, Mehmet Fatih

M.S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Halit Oğuztüzün

August 2007, 120 pages

One of the promises of High Level Architecture (HLA) is the reusability of simulation components. Although HLA supports reusability to some extent with mechanisms provided by Object Model Template (OMT), when the developer wants to use an existing federate application within another federation with a different Federation Object Model (FOM) problem arises. She usually has to modify the federate code and rebuilt it. There have been some attempts to solve this problem and they, in fact, accomplish this to some extent but usually they fall short of providing flexible but also a complete mapping mechanism. In this work, a model based approach that mainly focuses on Declaration, Object and Federation Management services is explored. The proposed approach makes use of Model Integrated Computing (MIC) and .NET 2.0 technologies by grouping federate transitioning activities into three well-defined phases, namely, modeling, automatic code generation and component generation. As a side product, a .NET 2.0 wrapper to Runtime Infrastructure (RTI) has been developed to help developers create IEEE 1516 compatible .NET 2.0 federates in a programming language independent way.

Keywords: Model-Based Approach, FOM Independence, HLA, .NET.

ÖZ

FEDERASYON NESNE MODELİ BAĞIMSIZLIĞI PROBLEMİNE MODEL TABANLI YAKLAŞIM

Uluat, Mehmet Fatih

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Assoc. Prof. Dr. Halit Oğuztüzün

Ağustos 2007, 120 sayfa

Yüksek Seviye Mimari (HLA)'nın vaad ettiği en önemli hususlardan birisi de simülasyon unsurlarının tekrar kullanılabilirliğidir. HLA tekrar kullanılabilirlik vaadini büyük bir ölçüde Nesne Yönetim Şablonu (OMT) ile sunulan mekanizmalar ile sağlamış olsa da, geliştirici daha önceden geliştirmiş olduğu bir federeyi farklı bir Federasyon Nesne Modeli (FOM)'a sahip bir federasyonda kullanmaya kalktığı zaman, FOM Bağımsızlığı problemi ortaya çıkmaktadır. Bu genellikle geliştiricinin federe kodunda değişiklik yapması veya federe yapısını tekrar oluşturmasını gerektirmektedir. Daha önce yapılan çalışmalarda bu problemin çözümüne yönelik çeşitli girişimlerde bulunulmuştur. Bu girişimlerin çoğu problemi bir noktaya kadar çözmüşlerdir, fakat sunulan bu yöntemler genel olarak esnek ve birbiri ile uyumlu tam bir çözüm sağlayamamaktadır. Bu çalışmada Tanımlama, Nesne ve Federasyon Yönetimi servisleri üzerine yoğunlaşan bir yaklaşım izlenmiştir. Sunulan yaklaşımda Model Bütünleşik Hesaplama (MIC) ve .NET 2.0 teknolojileri kullanılmış, uygulanan adımlar ve aktiviteler modelleme, otomatik kod ve bileşen üretme aşamaları altında toplanmıştır. Bu çalışmada ayrıca Çalışma Zamanı Altyapısı (RTI) için .NET 2.0 kabuğu geliştirilerek, kullanıcıların IEEE 1516 uyumlu .NET 2.0 federelerini programlama dilinden bağımsız bir şekilde geliştirmelerine olanak sağlanmıştır.

Anahtar Kelimeler: Model-Tabanlı Yaklaşım, FOM Bağımsızlığı HLA, .NET.

To My Parents

ACKNOWLEDGMENTS

The author wishes to gratefully thank his supervisor Assoc. Prof. Dr. Halit Oğuztüzün for his invaluable guidance, advice and encouragements for this research.

The author would also like to thank Assoc. Prof. Dr. Veysi İşler for his support.

The technical assistance of Ph. D. students Mehmet Adak, Gürkan Özhan and Okan Topçu are acknowledged.

The author would also like to thank his managers at ASELSAN, Mr. Sami Duman and Mr. Ersel Ercek for their support.

Finally, the author wishes his special thanks to his family for their patience, support and motivation.

TABLE OF CONTENTS

ABSTRACT iv
ÖZ v
ACKNOWLEDGMENTS vii
TABLE OF CONTENTS viii
LIST OF TABLES x
LIST OF FIGURES xi
LIST OF ABBREVIATIONS xiii
CHAPTER
1. INTRODUCTION 1
1.1 Related Work 4
1.2 Thesis Overview 9
2. HIGH LEVEL ARCHITECTURE 11
2.1 Short History of HLA 12
2.2 IEEE Std. 1516-2000:Framework and Rules 12
2.3 IEEE Std. 1516.1-2000:Federate Interface (FI) Specification ... 14
2.4 IEEE Std. 1516.2-2000:Object Model Template (OMT)Specification 16
3. MODEL DRIVEN SOFTWARE DEVELOPMENT 20
3.1 Model Driven Software Development (MDSD) 20
3.2 Model Driven Architecture (MDA) 21
3.3 Model Integrated Computing (MIC) 23
3.4 Metamodeling 25
3.5 Generic Modeling Environment (GME) 26
3.6 HLA Object Model Metamodel (HOMM) 28
4. .NET 2.0 TECHNOLOGY 35
4.1 .NET 2.0 Framework Overview 35
4.2 C++\CLI 39

4.3 Assemblies 39
5. MODEL BASED SOLUTION TO THE FOM INDEPENDENCE PROBLEM 41
5.1 Problem Definition 42
5.2 Three-Phased Solution 46
5.3 Case Study 75
6. DISCUSSION AND CONCLUSION 90
6.1 Achievements 90
6.2 Limitations of Current Work 91
6.3 Future Work 92
REFERENCES 93
APPNDICES
A. USER'S GUIDE 99
B. THE IMPLEMENTED RTI SERVICES FOR FIM 104
C. INTERPRETER SOFTWARE 105
D. RTIDOTNET1516 LIBRARY 108
E. AUTOMATICALLY GENERATED FILES 113

LIST OF TABLES

TABLES

Table 1.1 Object Class Structure Table 8
Table 3.1 Mapping to layers of OMG. 25
Table 3.2 Brief descriptions of GME modeling concepts 27
Table 3.3 OMTClass Model Attributes 32
Table 3.4 InteractionClass Model Attributes 32
Table 3.5 OMTAttribute Model Attributes 32
Table 3.6 Attribute Model Attributes 33
Table 3.7 Attribute Model Attributes 33
Table 3.8 Attribute Model Attributes (cont.) 34
Table 5.1 Three layers defined by BON2 [55] 69
Table 5.2 The Object Class and Attributes for Environment Federate 84
Table 5.3 The Object Class and Attributes for Aircraft Federate 85
Table 5.4 The Object Class and Attributes for Meteorology Federate 87
Table 5.5 The Object Class and Attributes for Target Federate 88
Table B.1 The implemented RTI services 104

LIST OF FIGURES

FIGURES

Figure 1-1 Relationship between FOM and SOM 3
Figure 1-2 The FOM-specific API approach illustration [9] 5
Figure 1-3 The FOM-configurable Fixed API approach [9] 6
Figure 2-1 The relationship of FI Specification, RTI, federates with each other.. 15
Figure 3-1 The illustration of basic principles of MDA 22
Figure 3-2 MIC Tools [46] 23
Figure 3-3 Constraint View of GME 28
Figure 3-4 GME class diagram of Federation Design Model [10] 30
Figure 3-5 The GME Object Models diagram [10] 31
Figure 4-1 Architecture defined by CLI [64] 36
Figure 4-2 Simple view of how .NET framework works [68] 38
Figure 5-1 Target and Source Federation/Federate relationship 43
Figure 5-2 FIP Dependency Levels 44
Figure 5-3 The three phases of proposed solution and their relationship 47
Figure 5-4 Overview of software components used in this study 48
Figure 5-5 The War Game Federation Design top level model 48
Figure 5-6 AttributeConverter Atom&other related metamodel elements 51
Figure 5-7 An example usage of AttributeConverter 52
Figure 5-8 ParameterConverter Atom&other related metamodel elements 54
Figure 5-9 Synchronizations paradigm sheet 55
Figure 5-10 An example usage of SynchronizationPointMapper 56
Figure 5-11 CoordinateConverter mapping 59
Figure 5-12 Publish/Unpublish services scenario with many-to-many formation 60
Figure 5-13 Register/Discover services scenario with many-to-many formation. 62
Figure 5-14 Update service scenario with many-to-many formation 64
Figure 5-15 A scenario for Create service 65

Figure 5-16 Constraints defined for FIM 67
Figure 5-17 Snapshot for Configuration Arguments..... 74
Figure 5-18 War Game Federation, C3 Federation, their members and their relationship with each other 76
Figure 5-19 Correspondence Models constructed for Environment & Meteorology Federates..... 76
Figure 5-20 Conversion Method for TemperatureConverter 77
Figure 5-21 Publish Directed Conversion Method for WindDirectionConverter. 77
Figure 5-22 Subscribe Directed Conversion Method for WindDirectionConverter 78
Figure 5-23 Conversion Method for WindSpeedConverter 79
Figure 5-24 Correspondence Models constructed for Aircraft&Target Federates 79
Figure 5-25 Publish Directed Conversion Method for CoordinateConverter 81
Figure 5-26 Subscribe Directed Conversion Method for CoordinateConverter ... 82
Figure 5-27 Environment Federate GUI 84
Figure 5-28 Aircraft Federate GUI..... 85
Figure 5-29 C3 Federation Design Model..... 86
Figure 5-30 Meteorology Federate GUI..... 87
Figure 5-31 Target Federate GUI..... 88
Figure 5-32 An example HLA Federation that shows possible scenarios 89
Figure A-1 Steps necessary to go related model where Correspondence Model constructed 100
Figure A-2 The Part Browser Window for Attributes..... 101
Figure A-3 The Object Inspector Window for Converter attributes 102
Figure A-4 The Control Panel of Interpreter..... 103
Figure C-1 Illustration of Interpreter software in modeling environment..... 105
Figure C-2 A screenshot of FOM Conversion Interpreter 107
Figure D-1 Snapshot for Configuration Arguments..... 110
Figure D-2 A code portion related with exceptions 111

LIST OF ABBREVIATIONS

Abbreviation or Symbol	Text
API	Application Programming Interface
BON	Builder Object Network
C3	Command, Control and Communication
CF-RFOM	Common Foundation Reference FOM
CIL	Common Intermediate Language
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CLS	Common Language Specification
CMDE	Correspondence Model Design Environment
COM	Component Object Model
CPU	Central Processing Unit
CTS	Common Type System
DIS	Distributed Interactive Simulation
DLL	Dynamic Link Library
DMSO	Defense Modeling and Simulation Office
DOD	Department of Defense
DSL	Domain Specific Language
DSME	Domain-Specific MIPS Environment
ECMA	European Computer Manufacturers Association
FCL	Framework Class Library
FCO	First Class Object
FDD	FOM Document Data
FEDEP	Federation Development&Execution Process
FIM	FOM Independence Metamodel
FIP	FOM Independence Problem

FOM……Federation Object Model

GME……. Generic Modeling Environment

GUI……... Graphical User Interface

HLA…….. High Level Architecture

HOMM……. HLA Object Model Metamodel

IEEE…….……Institute of Electrical and Electronics Engineers

JIT……Just-in-Time

JVM…….. Java Virtual Machine

LLDMS……Latitude Longitude Degrees Minutes Seconds

MDA……Model Driven Architecture

MDSD…….…….....Model Driven Software Development

MFC……..Microsoft Foundation Classes

MGRS…….....Military Grid Reference System

MIC…….…….....Model Integrated Computing

MIPS……Model Integrated Program Synthesis

MOF…….……Meta Object Facility

MOM…… Management Object Model

MON…….…… Meta Object Network

OCL……..... Object Constraint Language

OM…….……...Object Model

OMG…….....Object Management Group

OMT……. Object Model Template

PC……. Personnel Computer

PDM…….…Platform Definition Model

PIM……...… Platform Independent Model

PSM……..... Platform Specific Model

POC…….. Point of Contact

QVR Queries/Views/Transformations

RPR-FOM…….…...…Real-time Platform Reference Federation Object Model

RTI…….……...	RunTime Infrastructure
SISC…….…….....	Simulation Interoperability Standards Committee
SISO…….…….....	Simulation Interoperability Standards Organization
SOM…….…….	Simulation Object Model
TENA…….……...	Test Enabled Network Architecture
US…….…….....	United States
USDAT…….……	Under Secretary of Defense for Acquisition and Technology

CHAPTER 1

INTRODUCTION

High Level Architecture (HLA) proposed as a common language and an integrated software architecture that provides a general framework within which Modeling & Simulation (M&S) developers can structure and describe their distributed simulation applications [1]. After it is proposed at 1991 by United States (US) Department of Defense (DoD), it started to attract many attentions. The studies about HLA and community started to enlarge with this proposal. In 2001, HLA become an IEEE (Institute of Electrical and Electronic Engineers) standard as the IEEE Std. 1516, 1516.1, 1516.2, 1516.3 specifications, which are used extensively in this and similar studies [1], [2], [3], [4], [5]. In fact, it also inspires other new distributed simulation technologies like Test Enabled Network Architecture (TENA) which is another HLA-like standard [6].

The most important purposes of the HLA are reusability and interoperability that is provided by Object Model Template (OMT) [3] and Federate Interface Specification [2] respectively, which are two of three components of HLA. The other one is HLA rules, which is described in chapter 2. The OMT provides a standard mechanism to define and document the form, type and structure of data that will be shared among federates in federation. To be able to make object model reusable, OMT must include a minimum but sufficient degree of meta-level information in the object model description. The HLA Federate Interface Specification describes the runtime services offered to federates by the Run Time Infrastructure (RTI) [2], a software that implements this interface specification. It

provides services and functions, which are necessary to support an HLA compliant simulation.

The HLA defines the whole distributed simulation as federation with its members, which are called federate. Federates can be thought as the atomic executable units of HLA federation which are usually used to simulate a given function, monitoring, logging, visualization and for similar purposes. For example, a federate can be a tank simulation which simulates whole tank functionality for one federation or a federate can simulate only the turret control subsystem as a federate for one federation. The federation is the collection of such federates, to form a federation at least one federate is needed. The RTI is software which implements IEEE 1516.1 Federate Interface Specification, and, thus, provides necessary services for federates to communicate and share data with each other in a federation during execution time.

The HLA has been used by many developers in various kinds of distributed simulation projects more than eight years since it was first introduced at 1998 by US DoD as version 1.3 [7]. As HLA become more prevalent, some part of it gone under revision with some improvements and become an IEEE 1516 standard in 2000 [1], [2], [3], [4]. In fact there is an undergoing revision on IEEE 1516 [8]. Through this period, many federates and federations have been developed and some problems related with reusability arouse with the practical usage of HLA. Although reusability is one of the most important promises of HLA, how could it be possible to experience such problems? HLA provide reusability through the OMT Federation Object Model (FOM) and Simulation Object Model (SOM). FOM describe and define the data that will be shared among federates in the federation using Object Oriented fashion methods like inheritance. All federates are required to agree on this FOM. SOM describe and define the data specific for a federate and can also be provided to federation. The below Figure 1-1 shows the relation between an Airplane and Radar SOM and C3 (Command Control and Communication) FOM.

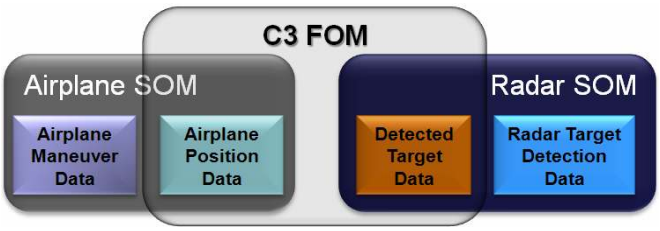

Figure 1-1 Relationship between FOM and SOM

As it can be seen from above Figure 1-1, FOM contains data that described in each federates' SOM. A FOM might contain all data defined in SOM which means that federate share all the data modeled in its SOM. A federate developer can reuse these models, while developing a new federate.

At this point, now let us go back to the reusability question. HLA, in fact provide mechanisms to developer for reusability through OMT and it accomplishes this while developing new federates, but when federate developer wants to use her already developed and well-defined federate within another federation that have different FOM, the FOM Independence Problem (FIP) comes up. The most obvious reason for this problem, discussed in [9], is the domain and the concepts represented with these federations or federates might change from one federation to another. The fidelity level of simulation and the need for usage of federates with global-wide federation which can cause standard conflicts are some other reasons for this problem.

Some studies have been performed to deal with this problem. Next section introduces two of these important studies by discussing what they bring and what they lack of. In this thesis, the HLA Object Model Metamodel (HOMM) [10], which is a metamodel to define OMT Models, is used; to make it useful for FIP, some additions have been made to create FOM Independence Metamodel (FIM).

The Generic Modeling Environment (GME) [11], which is used to define HOMM and FIM, employed as the Metamodeling and Modeling Environment. A .NET 2.0 RTI interface is provided to developers to develop HLA compatible applications in .NET 2.0 which is developed using C++\CLI programming language in Visual

Studio .NET 2005 [12]. In addition to use of it as an RTI .NET wrapper, it is also used for automatic code generation which is described in section 5.2.2. These approaches, technologies, tools and rationale for selecting these are discussed in forthcoming chapters.

This study proposes a model based solution for FIP using aforementioned technologies.

1.1 Related Work

In this section, two important studies which share similar objective with this study about FIP is discussed, what they provide and what they lack. There is also another approach which makes use of a base FOM to solve FIP. This approach is also discussed briefly by explaining an important example of related approach that is called the Real-time Platform Reference Federation Object Model (RPR FOM, pronounced “reaper fom”).

1.1.1 MÄK Technologies VR-Link Tool

One of these studies was carried out by MÄK Technologies [13]. In this study, the problem is briefly defined, two prevalent solutions mentioned and the solution they employ in their VR-Link tool is described [14]. It is stated in [9] that the solution for this problem is classified into two groups; the first one is “FOM-specific Code Generation Approach” and the second one is “FOM-configurable Fixed-API Middleware Approach”. The first approach in fact is the most obvious way in which such a way that developer develops an Application Program Interface (API) for a specific FOM. What these do is take FOM files, produce code specific to this FOM and then generate code that make use of these FOM for HLA specific calls. An example code that will be generated for an attribute in a given FOM which state status of number of bullets will be like below;

```
void RelatedHLAObjectClass::setNumberOfBullets(int iNumberOfbullets);
    int RelatedHLAObjectClass::getNumberOfBullets();
```

However, as stated before when a FOM changes, the API and depending on this the federate code needs to be modified and recompiled. Although this approach

might be useful for federations of whom FOM do not change frequently and could abstract details of HLA from developers, it will not be useful if FOM changes frequently. Although this approach reduces HLA developers programming burden, it does not provide a FOM independent solution as stated in [9], and it does not solve the FIP. This approach is illustrated in Figure 1-2.

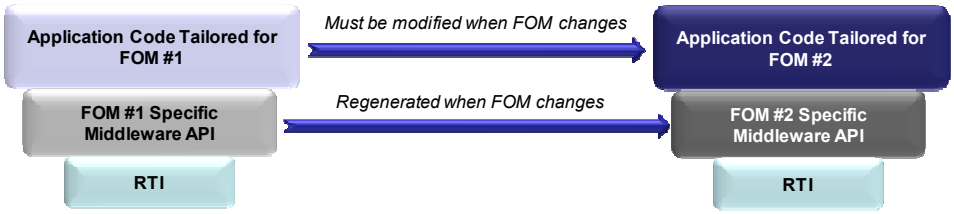

Figure 1-2 The FOM-specific API approach illustration [9]

This approach is used in many other commercial HLA products which make use of FOM to generate HLA code [15], [16], [17] as well.

The second approach mentioned looks more promising than the first one as it provides a mechanism for the developer to develop the federate code once and use it without any change for new federations. As stated in [9], in this approach a fixed FOM-Configurable API is provided to developer and when FOM changes a new FOM mapper is generated and used without changing federate code. Here the developer develops its federate code without considering future FOM changes, the rest is done by fixed API and FOM Mapper. The one key application here from FOM Independence point of view is FOM Mapper. The implementation provided by MÄK, uses a table of encoding, checking, and decoding functions, one set of functions for each attribute of each class. The mechanism applied by this implementation is summarized in [9] as;

> When generating an outgoing attribute update, the middleware toolkit asks the FOM Mapper for a checking function that checks whether the update condition holds for a particular attribute, and an encoding function that converts the attribute from the FOM-independent API's

representation to the current FOM's representation. When the middleware toolkit receives an incoming attribute update, it asks the FOM Mapper for a decoding function that does the opposite conversion.

The second approach is illustrated in

Figure 1-3 as below;

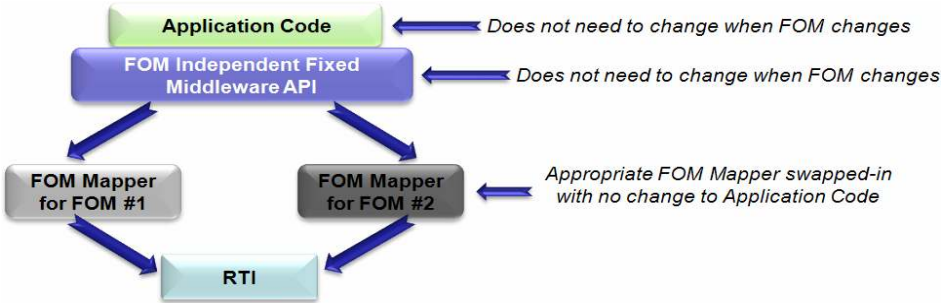

Figure 1-3 The FOM-configurable Fixed API approach [9]

This approach is used in MÄK's VR-Link product [14]. MÄK uses a propriety API for VR-Link to support other technologies besides HLA.

1.1.2 AEgis OMni Tool

The other study is provided by AEgis with OMni tool and it also targeted the same issue by employing an API [18], [19]. Different from FOM mapping approach that is mentioned in previous section, OMni provides developer a programming language for mapping called OMLink with its compiler. This language contains C++ like declarative atoms and SQL like procedural concepts. Below OMLink code portion shows an example mapping from [19] for "Detonate" and "Impact" which are stated as two different interactions that are used in different federations;

```
receive (Detonate(MunitionType Distance)) as (Impact(Damage))
{
  // Code to translate MunitionType & Distance to Impact.Damage
  // ...
}
```

They also provide a FOM mapper to use FOM/SOM elements and convert them to OMLink statements. Similar to shown receive statement in above code portion, there are three other statements for send, update and reflect. Similar to MÄK's VR-Link, OMni inspects and uses only update/receive/reflect/send services, the other services are not mentioned.

1.1.3 Real-time Platform Reference FOM (RPR-FOM)

A different approach to solve FIP is to agree on a common FOM and use it for all federates and federations like RPR FOM. As stated in [20], RPR-FOM was designed to organize the attributes and interactions of Distributed Interactive Simulation (DIS), which is another well-known IEEE distributed simulation standard [21], into a robust HLA object hierarchy. The motivation for developing such a design is listed as:

1. Support transition of legacy DIS systems to the HLA.
2. Enhance a-priori interoperability among RPR FOM users.
3. Support newly developed federates with similar requirements.

From the FOM Independence point of view the second and third items are apparently important to provide interoperability, but as discussed in the following section they can be limited.

The RPR FOM is an instance of a Common Foundation Reference FOM (CF-RFOM) as defined by the Simulation Interoperability Standards Organization's (SISO) Reference FOM Study Group [20]. A CF-RFOM is different from normal FOMs, because it refers to a notional FOM rather than an actual collection of federates. As stated in [20] the goal of a CF-RFOM is to enhance a-priori interoperability by specifying content standards for commonly used attributes and interactions. Federate developer build her Reference FOM using CF-RFOM according to her problem specific needs of federation and because federations that do not require interoperability beyond the basic level of the CF-RFOM can participate into federations, without software modification.

The RPR-FOM is developing in parallel to HLA, the Version 1.0 of the RPR FOM aims to provide an HLA conversion path for DIS capabilities as defined in IEEE 1278.1-1995 and it supports Version 1.3 of the HLA [20]. The Version 2.0 of the RPR FOM is planned to add the functionality of the IEEE 1278.1A-1998 standard [22] and also be compatible with the IEEE 1516 HLA standard [1]. In [20] it is stated that when transitioning of existing DIS functionality transition is completed, the RPR FOM Version 3.0 is planned to be release to capture new real-time simulation data exchange solutions compliant with IEEE 1516 standard. RPR-FOM – Object Class Structure Table is given in Table 1.1.

Table 1.1 Object Class Structure Table

Class 1	Class 2	Class 3	Class 4
BaseEntity	PhysicalEntity	Platform	Aircraft
			AmphibiousVehicle
			GroundVehicle
			Spacecraft
			SurfaceVessel
			SubmersibleVessel
			MultiDomainPlatform
		Lifeform	Human
			NonHuman
		Sensor	
		Radio	
		Munitions	
		CulturalFeature	
		Expendables	
		Supplies	
	EnvironmentalEntity		
EmbeddedSystem	Designator		
	EmitterSystem		
	RadioReceiver		
	RadioTransmitter		
EmitterBeam	RadarBeam		
	JammerBeam		

1.1.4 Discussion

When we look at the approaches discussed in [9], the most feasible solution seems to be the FOM-Configurable Fixed API approach. Although it solves FIP in many ways, it can compel developer to stick with a propriety API rather than HLA standard which increase time to learn related API like in VR-Link case. As seen

from the applied examples like VR-Link, usually tables are used for mapping which might limit its capability, configurability and understandability. Moreover, the mentioned usage of FOM Mapper and their middleware which make use of checking attributes with given functions at run-time might cause problems for mappings that involve more than one attributes owned by different object classes.

The approach proposed by AEgis solves the propriety API problem by providing an HLA like API which has similar function signatures as HLA standard. Although the OMni Link programming language is more intuitive than MÄK's table filling approach, as the previous approach the mapping mechanism employed by them is not so much developer friendly, even though a supporting FOM Mapper Graphical User Interface (GUI) is provided.

A rather different approach is explained through RPR-FOM. Although this might solve the problem to some extent for federations that uses RPR-FOM, in reality, there are various kinds of applications and specific concepts which are very difficult and sometimes impossible to capture them with only one FOM. In fact, for even more prevalent concepts; such as, geographical coordinates, which can have many different representations that resulted from standards or specific constraints. There are also some ongoing studies to enhance prior-FOM agreement for interoperability which share similar approach to [23], but it is not completed and available yet.

When all these studies are examined, it is seen that all of them solves FIP to some extent but usually they are lack of providing an easy, flexible but also a complete mapping mechanism which is in fact the most important motivation of this study.

1.2 Thesis Overview

The introduction chapter gives brief background information and mentions related studies about topic, the thesis continues with Chapter 2 which introduces HLA (IEEE 1516). The Model Driven Software development, GME and modified HOMM is explained in Chapter 3. The Chapter 4 is about .NET 2.0 and its features used in this study. Chapter 5 presents the contribution of thesis to FIP

with example federation that modeled and implemented in this approach. The summary and future works are given in Chapter 6. In Appendix A, a short user guide for constructing correspondence map in GME is described. The table of implemented services is given in Appendix B. The `Interpreter` software is described in Appendix C and `RTIDotNet1516` library is explained in Appendix D. Finally the details of source code generated in automatic code generation phase is given in Appendix E.

CHAPTER 2

HIGH LEVEL ARCHITECTURE

The HLA is the one of most significant distributed simulation technology and the specific domain that is used for this study. It is defined by IEEE 1516 for simulation developers to structure and define their applications using this as a common language and integrated software architecture. It is not merely an implementation, an API or similar software construct but a software architecture that defines some rules and protocols to compose independently developed simulations into one larger simulation with minimal effort and time [24]. A well known definition for software architecture is given by Shaw and Garlan [25];

> Abstractly, software architecture involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns.

Kuhl at [24] gave us mapping of element, interactions and pattern concepts of software architecture given in this definition to HLA standard which surely gave us a better understanding. In HLA world, the HLA federation elements (members) are federates, an RTI, and a common object model are defined by rules and interface specification. The interactions are defined between federates and the RTI, and between federates through RTI of whom data is defined by federation's object model. The allowed patterns of composition in the HLA are constrained by the rules and defined in the federate interface specification. Kuhl also illustrate some architectural styles that HLA exhibits in [24] which are Layered, Data Abstraction and Event-Based architecture. All these concepts are described in following sections in detail, now a brief history of HLA is given.

2.1 Short History of HLA

The HLA was first issued by the Defense Modeling and Simulation Office (DMSO) of US DoD, in order to support reuse and interoperability across the large numbers of different types of simulations and to reduce the cost of the projects [26]. The HLA Baseline Definition, HLA 1.0, was completed and approved by the Under Secretary of Defense for Acquisition and Technology (USDAT) in 1996. This approval not only defines a standard but also mandate all DoD simulations to use this standard. The other important release was happened in 1998 as HLA specification 1.3 which was also made publicly available. Then OMG [27] considered HLA as the Facility for Distributed Simulation Systems in 1998 and updated it in 2001 to reflect the changes resulting from commercial standardization of the specification. Finally, the HLA was approved as an open standard through the IEEE, namely IEEE Standard 1516, in September 2000. After this time, SISO [28] has also proposed some improvements for HLA IEEE 1516. In fact there is an undergoing revision on IEEE 1516 [8], [23] currently which is not resulted at the time of this thesis being written.

The full name of the standard is IEEE Std. 1516-2000 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA). It was prepared by the HLA Working Group, sponsored by the Simulation Interoperability Standards Committee (SISC) of the IEEE Computer Society. This standard consists of four related standards which are IEEE Std. 1516-2000: IEEE Standard for M&S HLA Framework and Rules [1], IEEE Std. 1516.1-2000: IEEE Standard for M&S HLA Federate Interface Specification [2], IEEE Std. 1516.2-2000: IEEE Standard for M&S HLA Object Model Template (OMT) Specification [3] and IEEE Std. 1516.3: IEEE Recommended Practice for HLA Federation Development and Execution Process (FEDEP) [4].

2.2 IEEE Std. 1516-2000: Framework and Rules

This standard gives an overview of the HLA, HLA concepts and defines a set of rules that apply to HLA federations and federates [1]. These are the rules that all

HLA compliant federates and federations should comply. There are ten rules, five of these are defined for federation, and the other five rules are defined for federates. The rules apply for federations are given with brief explanations below:

- "Federations shall have an HLA FOM, documented in accordance with the HLA OMT". This rule state an agreement on the information defined syntactically.
- "In a federation, all simulation-associated object instance representation shall be in federates, not in the RTI". RTI will be responsible for coordination and management of federation execution according to Interface Specification [2]. It will not hold any state information about federation or federate in itself which obviously break interoperability and reusability.
- "During a federation execution, all exchange of FOM data among joined federates shall occur via the RTI". This rule again defined to assure interoperability. Some possible unacceptable example for this rule can be a socket connection between two federates in addition to RTI.
- "During a federation execution, joined federates shall interact with the RTI in accordance with the HLA federate interface specification". The member federates should not only communicate through RTI but also comply with interface specification. This rule also ensure the successful execution of federates with different RTI implementations.
- "During a federation execution, an instance attribute shall be owned by at most one joined federate at any given time". This rule ensure one ownership for one object instance attribute which will be responsible from updating state of that attribute, but there are some services provided to transfer this ownership from one federate to other or perform similar actions during federation execution in Data Distribution Management services.

The rules for federates are as follows:

- "Federates shall have an HLA SOM, documented in accordance with the HLA OMT". As stated in federation rule 1, federates that want to comply with HLA should expose their minimal simulation functionality in their SOM.
- "Federates shall be able to update and/or reflect any instance attributes and send and/or receive interactions, as specified in their SOMs". This rule specifies that federate should be able to initiate appropriate behavior through RTI as long as it stick with its SOM.
- "Federates shall be able to transfer and/or accept ownership of instance attributes dynamically during a federation execution, as specified in their SOMs". Every federate should implement necessary ownership protocol through services that are defined in the interface specification, it is also necessary to state whether federate will transfer ownership during federation execution in its SOM.
- "Federates shall be able to vary the conditions under which they provide updates of instance attributes, as specified in their SOMs". As stated in standard, different federations may exhibit different conditions for update of instance attributes (e.g., specified update rate or threshold for update). The applicable conditions for update of specific instance attributes owned by a federate should be documented in federate's SOM.
- "Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation". This rule specify that each federate may use functions provided by RTI's Time Management Services to manage its logical time or may not use at all.

Most of the rules quoted above are addressing the FOM/SOM to enforce the reusability.

2.3 IEEE Std. 1516.1-2000: Federate Interface (FI) Specification

HLA interface specification describes the standard runtime services and interfaces offered to federates which are necessary to support an HLA compliant simulation and make it possible for federates to be able to interact and share data with other

federates in a distributed federation execution. Federate interface specification extends the HLA Rules and also describe the HLA concepts in detail. The software that implements and provide these services called RTI. This software allows federates to interact with each other through the implemented interface specification.

Physically, federates interact only with RTI, no other interaction mechanism with other federates is possible under standard. The HLA federate interface specification defines the services that the RTI software should provide to federates. Figure 2-1 shows relationship of Federate Interface Specification, RTI, federates with each other.

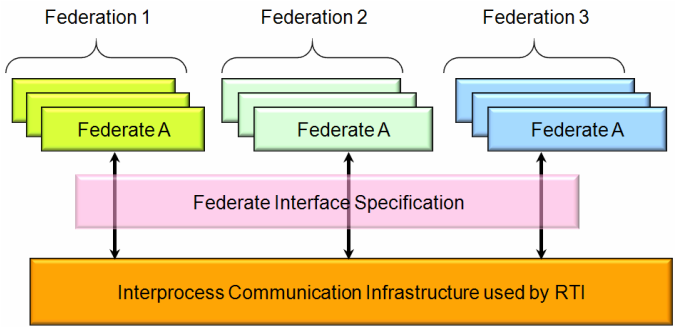

Figure 2-1 The relationship of FI Specification, RTI, federates with each other

The services defined by the interface specification are divided into seven service groups which are described as below;

- **Federation Management Services**: This group of services is responsible from creation, modification, deletion and dynamic control of a federation execution. It also contain services for federate to join or resign from a federation and for synchronization point management services.
- **Declaration Management Services**: This group of services is responsible from services that are used by federates to declare their interest to an object class attribute or an interaction class. These are also used to declare

intention to generate information through publish/unpublish and subscribe/unsubscribe services. This group of services is also handled in this study.

- **Object Management Services**: This group of services is mainly responsible from exchange of data through registration, modification, and deletion of object instances and the sending and receiving of interactions. This group also includes services that deal with how data are transported.
- **Ownership Management Services**: This group of services is responsible from transferring ownership of instance attributes among joined federates during federation execution. This group of services may not be used at all if no ownership transfer related capability is need for federation
- **Time Management Services**: This group of services is responsible from coordination of logical time among federates during federation execution. Similar to Ownership Management services this group of services may not be used by federates.
- **Data Distribution Management Services**: This group of services is responsible from providing information on data relevance at different levels and allows refining the data requirements.
- **Support Services**: This group of services includes miscellaneous services like setting advisory switches, manipulating regions, or getting attribute/interaction handles and names.

This specification also introduces Management Object Model (MOM) which is not studied in this study.

2.4 IEEE Std. 1516.2-2000: Object Model Template (OMT) Specification

The OMT Specification provides a standard mechanism to define and document the form, type and structure of data that will be shared among federates through federation. To be able to make object model reusable, OMT must include a minimum but sufficient degree of meta-level information in the object model

description. Moreover in specification [3], it is stated that such specification does not only provide, a commonly understood mechanism for specifying the exchange of data but also a standard mechanism for describing the capabilities of potential federation members and facilitation of common tool sets for development of HLA object models.

OMT standard introduces two types of object models: HLA Simulation Object Model (SOM) and Federation Object Model (FOM).

The SOM describe and define the information that a federate can provide to other HLA federations and the information that it can receive from other federates in HLA federations. It is usually a specification of federate specific capabilities that an individual simulation that modeled by federate could provide to HLA federations, in other words it focuses on federate's internal operation. It is defined for each federate and can also be used to determine the suitability of federates for participation in a federation.

The FOM describe and define the data that will be shared among federates in HLA federation using Object Oriented fashion and enforce all federates to agree on this FOM. FOM is defined for each federation and usually provided to each federate execution physically through FOM Document Data (FDD) files. This FDD files contains information derived from the FOM and used by the RTI at runtime in XML format.

The OMT Specification presents object models in OMT tabular format. The OMT consists of fourteen components that presented as tables:

- **Object Model Identification Table**: This table contains information about the name, type, version, modification date, purpose, application domain, sponsor, POC (point of contact) name, POC organization, POC telephone, POC E-mail address, references and some other information about the object model.
- **Object Class Structure Table**: Object class structure table define all federate or federation object classes and their class-subclass relationship.

An object class can be considered as a collection of objects with certain characteristics or attributes in common [3]. Object Class Structure Table also contains attributes publishing/subscribing behavior with “P”, “S” or “PS” markings. “P” is used for publish behavior, “S” is used for subscribe behavior and “PS” for attributes which exhibits both of these behaviors. There is also “N/A” for attributes which do not exhibit any of these behaviors.

- **Interaction Class Structure Table**: Interaction class structure table contains all federate or federation interaction classes and their class-subclass relationships like in Object Class Structure Table. This table also contains interactions’ publishing/subscribing behavior with “P”, “S” or “PS” markings.
- **Attribute Table**: This table contains features of object attributes in a federate or federation.
- **Parameter Table**: This table contains features of interaction parameters in a federate or federation.
- **Dimension Table**: This component specifies the dimensions that are defined by `Attributes` and `Interactions` to filter the ones out of given dimensions at run-time.
- **Time Representation Table**: This table determine the usage of time stamps and look ahead characteristics of both federates and federations.
- **User-supplied Tag Table**: As stated in [3], federates can supply tags with certain of the HLA services to provide additional coordination and control over these services. This table defines these tags.
- **Synchronization Table**: This table defines the representation and data types used in HLA synchronization services.
- **Transportation Type Table**: This table describes mechanisms used for the transportation of data.

- **Switches Table**: This table contains initial settings for some parameters defined in federate interface specification.
- **Datatype Tables**: Basic Data Representation Table, Simple Datatype Table, Enumerated Datatype Table, Fixed Record Datatype Table, Array Datatype Table, and Variant Record Datatype Table specify the details of data representation in the object model.
- **Notes Table**: Additional information can be added to any element of the object model. This table is used for this purpose.
- **FOM/SOM Lexicon**: This table contains descriptive definitions for all of the objects, attributes, interactions, and parameters used in the HLA object model.

As described in the next chapter, these tables are provided to developer through the GME modeling environment defined by HOMM.

The detail of FEDEP is not given here as it is not directly related with FIP. More information about High Level Architecture can be found in [7], [24].

CHAPTER 3

MODEL DRIVEN SOFTWARE DEVELOPMENT

This chapter provides information about Model Driven Software Development (MDSD) and related technologies which are the key technologies behind this thesis. HLA Object Model Metamodel (HOMM) which is taken as a base for this study will also be described. The important technologies and tools that are used in study are Model Driven Architecture (MDA), Domain Specific (DS) MDA which is also known as Model Integrated Computing, Metamodeling and GME. These are described in following sections, but before that little background information about Model Driven Software Development and its role in Modeling and Simulation (M&S) will be given.

3.1 Model Driven Software Development (MDSD)

The first thing that needs to be described for MDSD is surely the concept of model. Although there are many definitions for model, the one that is provided from [29] is given below;

> A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system.

MDSD [30] is the one of the latest popular approach in software development which aims developing software from higher abstraction levels like from domain-specific models and make uses of the models as first class entities like source code. In fact, many attempts have been done till MDSD to increase abstraction level especially in languages, platforms and tools.

Although all of these seem to solve problems initially, the problems arouse again after using these over time like in the middleware approach. The middleware

approach provides a software layer that provides some common services for software components or applications from different platforms, operating systems. As middleware approach becomes common among software developers, many middleware platforms came out, moreover these middleware standards also changed in time. The cost of porting an application from one middleware to other is also requires high cost even the business logic does not change. Consequently it is also seen unrealistic to standardize on a single middleware platform.

After all these attempts, MDSD aim to separate Computing, i.e. Solution Domain from Problem Domain while increasing abstraction level and make use of models as the first class entities in software development process rather than a blueprint for documentation. In MDSD, models not used merely for documentation, they are considered equal to source code as their implementation is automated.

The most obvious promise of MDSD is the increase in productivity. The other important benefits are increasing development speed, managing complexity through abstraction, increasing portability, reusability and interoperability, and automating software construction with no or minimum coding.

As MDSD become popular in Software Development world, some attempts to apply this approach through MDA to HLA compliant distributed simulations has been made like [31], [32], [33], [34] to show the benefits of employing this approach.

For this study, the models are FOM, SOM and `Correspondence Models`, the system is HLA federation and goal is to solve FIP using these models. Moreover, HOMM [10], described in section 3.6, is used for FIP with some additions to this metamodel.

3.2 Model Driven Architecture (MDA)

Although the HOMM and this study mostly use MIC, for the sake of completeness, background information about MDA will also be given here.

The Model Driven Architecture (MDA) [35] is defined by the Object Management Group (OMG) [27] to achieve portability, interoperability and reusability to integrate distributed applications by focusing on the importance of models in the software development process [36]. The primary focus of MDA is the on the functional and behavioral aspects of a distributed application or system, not the technology in which it will be developed [37]. MDA achieved this by separating these two and define them as Platform Independent Model (PIM) and Platform Specific Model (PSM). First system functionality is defined by PIM, then by using a Platform Definition Model (PDM), which define target model, in collaboration with transformations the PIM is transformed to PSM and finally executable code is generated again using transformations for corresponding platform and Platform Models. The basic principles of MDA can be seen in Figure 3-1.

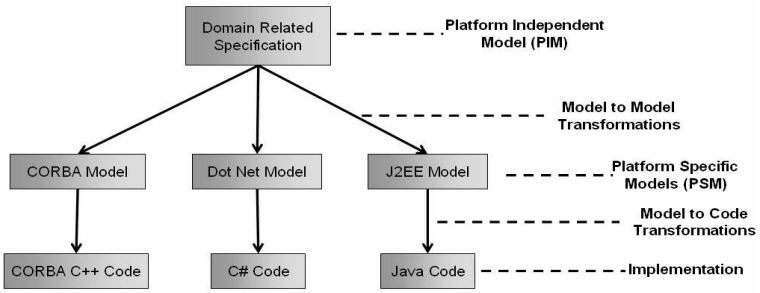

Figure 3-1 The illustration of basic principles of MDA

It is also worth to mention how new models can be obtained by using the existing ones before closing this section. Although there are various ways of obtaining new models, the model transformation is the most prevalent way of obtaining new models from old ones in model driven approaches. There are two major categories of transformations which are Model to Model and Model to Text or Code as illustrated in Figure 3-1 [38]. In addition to these two categories, other transformation categorizations and approaches belonging to these categories can be seen in related studies like [39], [40], [41]. As the interest for model

transformation increase and the importance of it become more apparent, the OMG defined QVT (Queries/Views/Transformations) as a standard for model transformation which can be obtained from [42].

In this study, the model linking and model transformations are heavily used. The model linking is used in construction of `Correspondence Model` among two different FOMs to show relation of `Attributes/Parameters` among source and target federation. Moreover, the automatic code generation phase use Model-to-Code Transformation to generate .NET compatible source code from `Correspondence Model` and FOMs. Chapter 5 describes these in detail.

3.3 Model Integrated Computing (MIC)

Many model driven approaches have appeared since MDSD first introduced like MDA and Software Factories [43]. Model integrated computing is one of these methodologies for developing domain-specific software that uses MDA concepts and metamodeling approach [44]. As mentioned, it is also known as Domain Specific (DS) MDA which is proposed to be an effective and efficient way of developing large-scale, domain-specific software. The tools that will be used in MIC are introduced by Karsai and his colleagues in [45] and [46] which are also shown as in Figure 3-2.

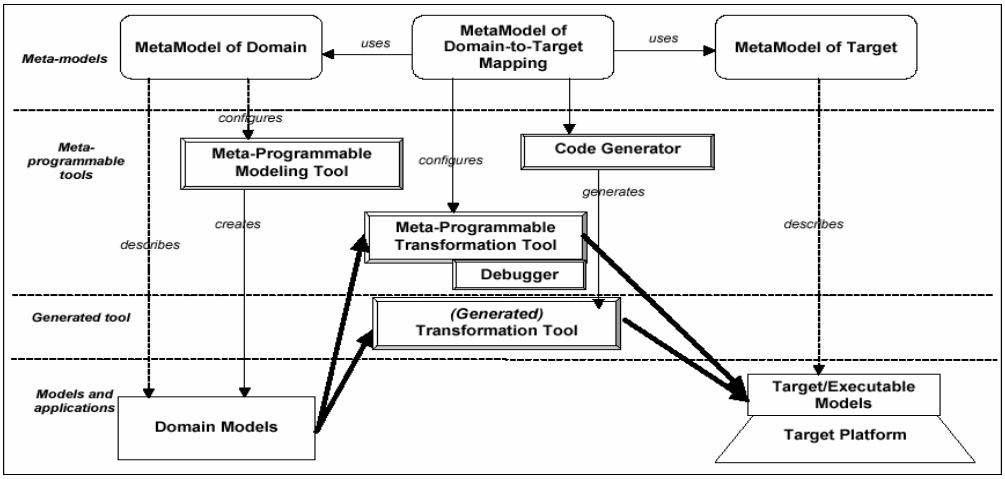

Figure 3-2 MIC Tools [46]

Before describing details of MIC process, it is beneficial to describe frequently used concepts like domain.

Domain can be described as a collection of entities that share the same characteristic or exhibit similar functionality [47]. Domain Specific Modeling is a software engineering methodology which models a system using the common terminology and concepts that are obtained from a domain analysis [48]. These terminology and concepts are usually belongs to problem domain rather than solution domain like programming language constructs. To represent and implement these concepts and terminology Domain Specific Language (DSL) are used. A DSL definition from [49] is given below;

> A DSL can be viewed as a programming language dedicated to a particular domain or problem. It provides appropriate built-in abstractions and notations; it is usually small, more declarative than imperative, less expressive than general-purpose language.

MIC let generation of applications from models by using customized domain-specific Model Integrated Program Synthesis (MIPS) environments. To do this, first DSL is defined formally for related domain, then a meta-level translation performed using this DSL to synthesize the Domain-Specific MIPS Environment (DSME) [50] from this metamodel. After creation of DSME, it can be used to create different domain specific models. These models are then used by model interpreters to perform semantic translations to generate executable models or programs [44].

As stated in [44], a MIPS environment operates according to a modeling paradigm which is a set of requirements that define the way systems within the domain are to be modeled. In other words, it defines the language for modeling systems in the domain. The modeling paradigm is then captured in the form of formal modeling language specifications called a metamodel which is also used commonly in other MDSD approaches. MIC applies this metamodel based approach to domain specific applications which are HLA applications in our case.

The HOMM study applies MIC for distributed simulations domain through HLA and in our study we use this approach to solve FIP.

3.4 Metamodeling

In previous sections, it is stated that modeling describes the concepts and terminology of a domain through a modeling language. Similarly, it can be said that metamodeling allows us to model the modeling language. It is also stated that model is an abstraction or simplification of a system in the real world, and metamodel is a higher abstraction that focus on features of the model itself. It can be considered as the language for expressing a model that describes relevant concepts of a domain. There are many benefits of metamodel. With metamodels, domain specific modeling become possible, models are validated against the constraints defined in the metamodel level, model transformations can be generalized through metamodel level rules and finally automatic code generation can be performed through templates that refer to the metamodel. There also exist Meta-metamodels which are used to describe the language of meta-models. Meta-metamodels are also important for defining languages and tool integration. The most important difference between modeling and metamodeling is the level of abstraction, the rest of these two activities are very similar.

Modeling, metamodeling, and meta-metamodeling languages and related actives are defined as four-layer metamodeling architecture by OMG [27], [51]. Table 3.1 shows the metamodeling layers for this study which mapped to OMG's four-layer framework.

Table 3.1 Mapping to layers of OMG.

Framework Model	Mapped Model
Metamodel (M3 Layer)	GME Metamodel
Metamodel (M2 Layer)	FOM Metamodel (FIM)
Model (M1 Layer)	HLA Model
Run-time instance (M0 Layer)	HLA IEEE 1516 Compliant .NET 2.0 Application

3.5 Generic Modeling Environment (GME)

For all modeling and metamodeling activities, Generic Modeling Environment (GME) which is an open-source meta-programmable modeling tool developed by Vanderbilt University [11] is used. The motivation for choosing GME as tool can be listed as below which are also mentioned in [10];

- It provides generic modeling primitives [52], which are necessary to create the domain-specific modeling concepts through meta-modeling.
- GME paradigms are generated from formal modeling environment specifications like stated in MIPS at section 3.3.
- In addition to generic metamodeling primitives, it also provides an environment containing all of the modeling elements and valid relationships that can be constructed in a specific domain, after a modeling paradigm is defined.
- It contains some integrated model interpreters through plug-in architecture that perform translation and analysis of models and provides defining new interpreters.
- The models are formed as graphical, multi-aspect, attributed entity-relationship diagrams. The semantics behind the model is determined during the model interpretation process which correspond to automatic code generation phase for our study.
- It supports multiple paradigms and enables meta-model composition.
- With Windows based and well organized graphical user interfaces used make it user friendly and easy to use.
- The tool is open source which can be used for academic purposes and have strong community support.
- It contains a constraint manager which is compliant with the Object Constraint Language (OCL) 1.4 specification [53] and its metamodel is implemented using Meta Object Facility (MOF) 1.4 specification [54].

The details of GME architecture can be found in [55], [56]. Here some important modeling concepts that are used in this study are explained. Some important modeling concepts and their brief descriptions are given in Table 3.2.

Table 3.2 Brief descriptions of GME modeling concepts

Modeling Concept	**Definition**
Project	The root container class.
Folder	Containers that help to organize models.
FCO	First-class objects which must be abstract but can serve as the base type of an element of any other stereotype.
Model	Compound objects that can contain model elements.
Atom	Atomic objects which are not containing other model elements but have attributes.
Set	Specify a relationship among a group of objects whose parent objects are same and visible in the same aspect.
Connection	Used to construct a relationship between two objects.
Reference	Make other objects except connection to be able to be used in other models.
Attribute	Property of an object which is expressed in text.
Aspect	Construct that provides logical visibility partitioning to present different views which are explained in below.
Constraint	Construct used to check correctness of a model using OCL like expressions.

As mentioned with Aspect concept, GME provides developer four different views which are classes, visualization, constraints and attributes. The most frequently used view is Classes View which allows developers to define its domain specific models with models, references, atoms and other elements. Creation of aspects and relating the model elements with these aspects is done in Visualization View. Constrains, constraint expression and relating these with model elements are performed in Constraints View. Finally, attributes for modeling elements are defined in Attributes View Figure 3-3 shows Constraint View of GME.

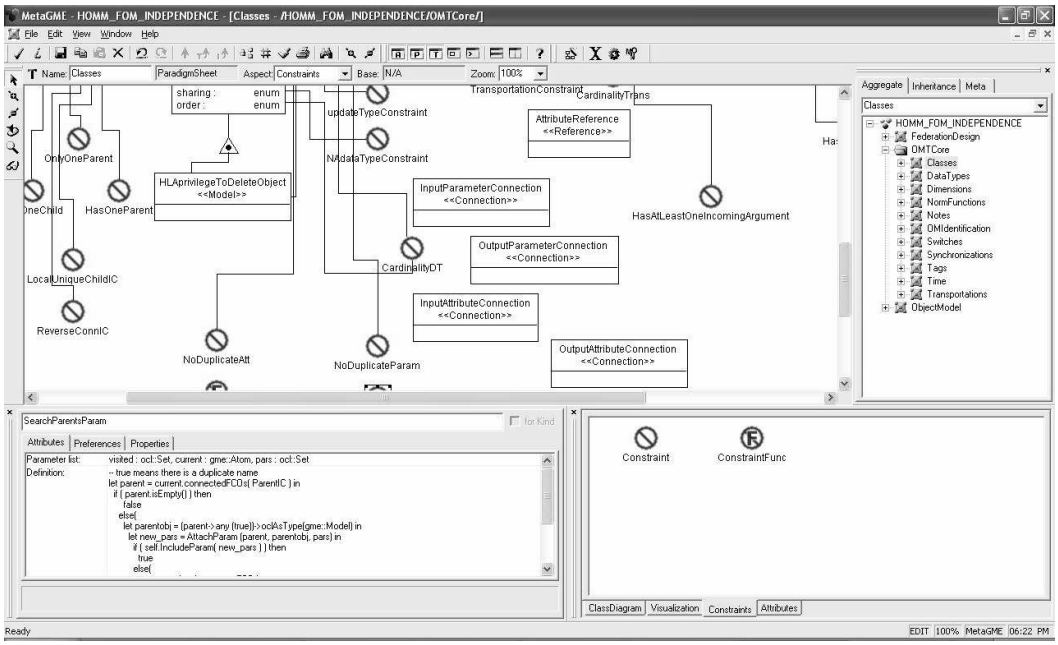

Figure 3-3 Constraint View of GME

3.6 HLA Object Model Metamodel (HOMM)

To solve FIP in HLA by using model driven approach, first of all a modeling capability for HLA domain is needed. The HLA Object Model Metamodel (HOMM) is chosen for modeling activities in this study and some additions have been made to it to use it in this study which is described in the next section [10]. The full compliance with IEEE 1516 HLA OMT and provided IEEE HLA Defaults library, which is extensively used in automatic code generation phase are the most important reasons for HOMM usage.

In addition to these, the HLA specific modeling environment created by GME tool which also used for defining HOMM is plays a significant role in decision. It does not only provide a more understandable and user friendly design environment than tabular or text based data modeling, but also an easily configurable MDA compliant HLA modeling tool for this study.

The HOMM is composed of Object Model (OM) paradigm sheet, Federation Design paradigm sheet and OMT Core folder. Paradigm sheets are used for

separation in metamodel. The definitions for classes, data types, dimensions, switches, synchronization points, user-supplied tags, time representations and transportations are included in OMT Core folder and separate paradigm sheets are defined for each of them. The OM paradigm sheet provides a metamodel for HLA OM which includes three types of object models, FOM, SOM and Other. In addition to HLA OM, another metamodel is created as Federation Design paradigm to form a high level modeling for federation federates and necessary modeling entities to connect them with related FOM's or SOM's.

Some rules made effective on whole metamodel in HOMM [10]. They also followed in developed metamodel with some additions. All these constraints and rules are written in OCL [53]. For this study, some additions have been made to these rules, some of which are defined in OCL as HOMM and some others are defined programmatically to be used in the beginning of automatic code generation phase.

Next sub-sections explain Federation Design model, Object Model and OMT Core folders in detail.

3.6.1 Federation Design Model

As stated in [10] Federation Design Model provides a higher level interface for modeling federates, federation and their connection with related FOMs or SOMs. This model can contain more than one federates and SOMs but there should be only one federation and one FOM. The connection with FOMs and SOMs are done through reference entity which is described in previous section. "MemberOf" relationship is used to define connection between federations and federates. The Figure 3-4 shows the GME class diagram of Federation Design Model.

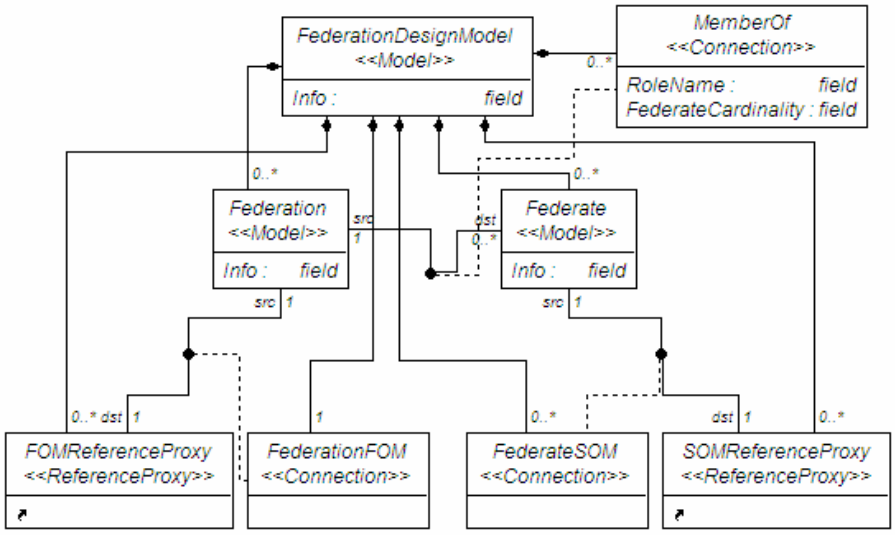

Figure 3-4 GME class diagram of Federation Design Model [10]

This model is not used in this study, but not excluded from metamodel also and preserved to be used for illustrative purposes.

3.6.2 Object Model

The Object Model (OM) paradigm sheet includes the main diagram for object models. As stated before, there are three types of object models, which are FOM, SOM and Other. FOM and SOM are correspondence of HLA object models that defined in HLA OMT specification. The "Other" type is provides as template for temporary object models which can not be included in Federation Design model [10]. The GME diagram of OM is given in Figure 3-5. As it can be seen in figure, Object Model class is the parent of FOM, SOM and Other. The parent-child relationship, which is similar to Object Oriented inheritance, is illustrated by a special triangle operator. There are also five aspects defined to model related OMT components which are Classes, User-Supplied Tags, Synchronization, Switches and Time Representation.

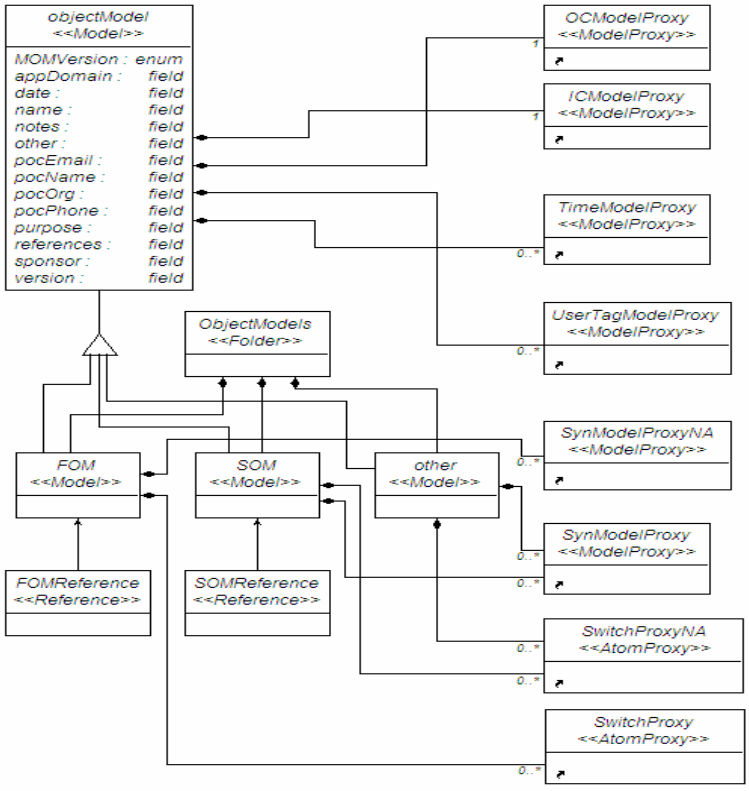

Figure 3-5 The GME Object Models diagram [10]

3.6.3 OMT Core Elements

OMT Core folder provides OMT Core elements which are necessary to model an object model [10]. It contains classes, data types, dimensions, normalization functions, notes, switches, synchronization points, user supplied tags, time representations and transportations models as stated before. Here brief descriptions of elements which are heavily used in our study is given, [10] can be consulted for detailed descriptions.

3.6.3.1 OMT Core Elements

The classes' paradigm sheet provides object class, interaction class, attribute, and parameter definitions for the HLA object models. These elements used to define object class structure table, interaction class structure table, attribute table, and parameter table which are specified in HLA OMT specification. Moreover, most of the elements have attributes. The attributes for OMTClass, InteractionClass,

OMTAttribute and Parameter model are given with their descriptions in Table 3.3, Table 3.4, Table 3.5 and Table 3.6. The attributes for ObjectClass and Parameters are inherited from their parents so they are not given.

Table 3.3 OMTClass Model Attributes

Attribute Name	**Definition**
Sharing	Publication (“P”) / Subscription (“S”) capabilities. The valid inputs are “P”, “S”, “PS”, “N (Netiher)”.
Definition	Information about the class.
Semantics	Semantics for the class.
Notes	Additional user notes.

Table 3.4 InteractionClass Model Attributes

Attribute Name	**Definition**
Order	Specifies the order of delivery. The valid inputs are “Receive” and “TimeStamp”.
Dimension	Available dimension.
Transportation	Specifies the type of transportation. “HLAbestEffort” and “HLAreliable” are valid inputs. These are provided through IEEE default library.

Table 3.5 OMTAttribute Model Attributes

Attribute Name	**Definition**
Datatype	The datatype of the attribute.
Definition	Information about the attribute or parameter.
Semantics	Semantics for the attribute or parameter.
Notes	Additional user notes.

Table 3.6 Attribute Model Attributes

Attribute Name	Definition
Dimension	Available dimension.
Transportation	Specifies the type of transportation.
UpdateType	Policy for updating instance of the class attributes. The valid inputs are “Static”, “Periodic”, “Conditional” & “N/A”.
UpdateCondition	Expanded and explained policies for updating an instance of the class attribute.
DivestAcquire	Indicates whether ownership of an instance of the class attribute can be divested or acquired. The valid inputs are “D (Divest)”, “A (Acquire)”, “N (NoTransfer)” and “DA (DivestAcquire)”.
Sharing	Same as Table 3.3 sharing description.

3.6.3.2 Synchronizations

This diagram is provided to support OMT synchronization points specified in section 2.3.

3.6.3.3 Data types

The data types are defined as shared model and used through all models like object class attributes, interaction class parameters, dimensions, time representations, user-supplied tags, and synchronization points. These data types are used in automatic code generation phase to generate data type in .NET to be used for conversion purposes. Table 3.7 gives these data type elements with brief descriptions.

Table 3.7 Attribute Model Attributes

Element Name	Definition
DatatypeModel	Abstract model stands on the top of other data types.
basicData	Forms the basis of the data types, not used directly (e.g. HLAfloat32LE, HLAinteger32BE).
simpleData	Used to describe simple, scalar data items (e.g. Integer, Float).
enumeratedData	Used to describe data elements that can take on a finite discrete set of possible values (e.g. HLAboolean).

Table 3.8 Attribute Model Attributes (cont.)

Element Name	Definition
arrayData	Used to describe indexed homogenous collections (e.g. HLAASCIIstring).
fixedRecordData	Used to describe heterogeneous collections of types known as records or structures. The fields can be defined in other types such as simple data types, fixed records, arrays, enumerations, or variant records (e.g. Vec3f, GPSData).
variantRecordData	Used to describe discriminated unions of types known as variant or choice records.

A previously defined set of basic data representations, predefined simple, enumerated and predefined array data types are defined in IEEE default library and provided to developer to be used in creation of models. These libraries are defined by developer and attached to the model if needed. Moreover some default transportations for HLA is also modeled provided with HOMM.

In HOMM there is also a metamodel for Management Object Model (MOM) which is defined in federate interface specification and provides additional facilities for access to RTI services during federation execution. In this study, MOM is not used, but it is left in this metamodel also as Federation Design Model to be able to be used by developers.

In this chapter, information about HOMM that used as base for this study and other technologies that made use in this study is described. The detail of metamodel for this study is given in chapter 5.

CHAPTER 4

.NET 2.0 TECHNOLOGY

The solution proposed for HLA FIP employs Model Based approach mentioned above and .NET 2.0 technologies. Note however, that this model-based approach, by its very nature, does not rely on any particular implementation technology, be it .NET or J2EE. When it comes to demonstrate the approach, however, a particular implementation technology has to be selected and .NET 2.0 is selected.

The most important reason to choose this technology is the lack of extensive HLA usage with .NET 2.0. There are many examples of Java usage which is very similar to .NET technology but we encounter with very few example studies that have been made in HLA world with .NET [57]. In addition to this, the provided programming languages and platform independent infrastructure, wide support of libraries, and a new programming language which let developers to use C++ programming language power with .NET 2.0 technology, C++\CLI [58], are other important reasons for our choice.

In this chapter, brief information about .NET 2.0 and related technologies that are employed in study are given. In the following sections .NET 2.0, programming language and platform independency, C++\CLI, assemblies and different .NET implementations are described.

4.1 .NET 2.0 Framework Overview

The motivation behind.NET 2.0 is providing an integrated environment for developing and executing applications on the Internet, on desktop PCs and on other platforms like pocket PC and smart phones easily. The .NET framework is separated into two parts to achieve these objectives; the Framework Class Library

(FCL) and the Common Language Runtime (CLR). Before describing these two, it is better to define Common Language Infrastructure (CLI) first.

To provide a portable environment that can be hosted by any operating system the Common Language Infrastructure is developed by Microsoft as an open specification and also becomes an International Standard Organization (ISO) European Computer Manufacturers Association (ECMA) ECMA-335 standard [59] that describes the executable code and runtime environment that form the core of the Microsoft .NET framework. This specification defines an environment that allows multiple high-level programming languages to be used on different platforms without being rewritten for specific architectures and in fact there exist other implementations of CLI for various platforms and operating systems (OS) in addition to Microsoft implementation for Windows based PC's, such as the Mono Project [60], DotGNU Portable .NET [61] and, .NET Compact Framework [62]. A list of .NET compatible programming languages can be found at [63].

As stated in [64], the CLI describes a platform-independent virtual code execution environment. The most important parts of the standard are the definition for a Common Intermediate Language (CIL) which must be generated by CLI compliant compilers and a type system, Common Type System (CTS) that defines the data types supported by any compliant language. The Figure 4-1 shows the architecture defined by CLI.

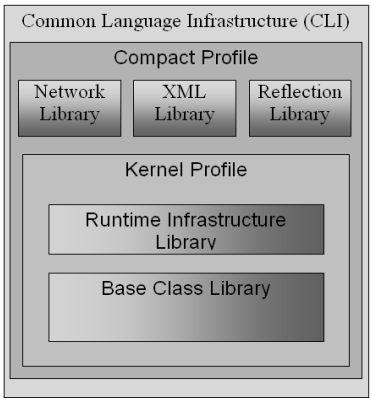

Figure 4-1 Architecture defined by CLI [64]

As illustrated in Figure 4.1, the CLI defines two profiles; the minimal conforming implementation of the CLI is called Kernel Profile, the profile that contains additional features useful for applications targeting a more resource-rich set of devices is called Compact Profile [65]. The details of these profiles can be found in [65].

As mentioned above CLI also define the CTS, which is an integral part of the CLR and provides a base set of data types for each language that runs on the .NET platform.

There exists a more restricted specification for programming language interoperability which is called Common Language Specification (CLS). As stated in [64], CTS itself is not enough to make languages comply with each other, and CLS is provided to solve this issue by providing minimal features for compilers that targeting CLR and language interoperability.

As stated in the beginning there are two parts of .NET framework. The first part is Framework Class Library (FCL) which is a collection of classes and other necessary types (enumerations, structures, and interfaces) that are available to the managed code. For more detailed information about these namespaces and classes [66] can be consulted.

The other and more important part is CLR which is simply as stated in [67], the virtual machine component of Microsoft .NET analogue to Java Virtual Machine (JVM) in Java. It coordinates the entire life cycle of a .NET application like locating code, compiling it, loading associated classes, managing its execution, and ensuring automatic memory management and security. In addition to this, it provides support for cross-language integration which permits code generated by different programming languages to interact seamlessly.

Now how .NET works is described from developer perspective. Developer develops its application in CLI compliant programming language such as C# or C++\CLI without focusing on specific Central Processing Unit (CPU) or OS.

Then a CLR compliant compiler compiles this code into managed code which is also known as CIL or MSIL for Microsoft Compilers. It also generates metadata to be embedded with code which contains information about the content of the code. CIL is analogue to byte code which is the code generated by Java compilers. This code is not an executable code, it is stored in an .EXE or .DLL file. When related code is executed, CLR's Just-in-Time (JIT) compiler converts this CIL code into the code that native to the OS and CPU.

The CIL here is important in such a way that it is the key construct that meets .NET's programming language independency objective. Due to CIL, CLR do not need to know the programming language that application is developed. Another important objective of .NET is platform independency. It is achieved by use of JIT compiler in such a way that when CIL is produced, it can be run on any other platform that has its own .NET framework and a JIT compiler that generate machine code specific for that platform. The Figure 4-2 shows a simple view of how .NET framework works.

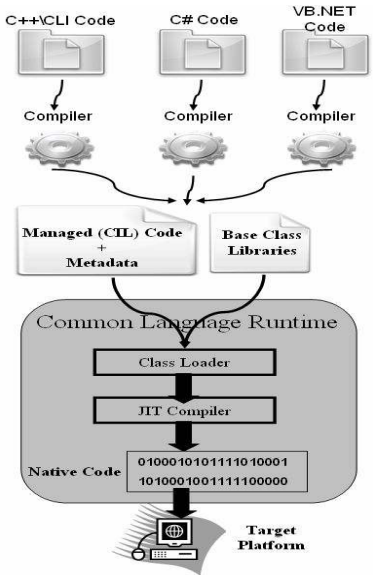

Figure 4-2 Simple view of how .NET framework works [68]

This whole process like in Java comes with an overhead of JIT compilation processes. Although it is argued that this difference is not too much, this could be a problem for critical cases. To avoid this overhead, tools like Native Image Generator can be used. This tool can skip the whole CIL code generation phase and convert whole code directly into native code for specific CPU before runtime [69] which accelerate the application with respect to normal usage.

4.2 C++\CLI

The C++\CLI is Microsoft's new programming language that comes with .NET 2.0 which let developers to use C++ programming language power with managed code technology. C++\CLI is also under process of being standardizes as ECMA standard [70]. In this study C++\CLI is used for the development of `RTIDotNet1516` library. The automatically generated code is in C++\CLI as well.

After deciding to implement the solution in .NET, it is needed to use .NET 2.0 compatible RTI software. Although there are many Java bindings for RTI, we only came across with [57] which are developed for .NET 1.1. So it is decided to develop a .NET 2.0 compatible RTI wrapper which means that all types, classes and methods should be ported to .NET 2.0. Till C++\CLI, the only ways to this use of marshalling related operations, P/Invoke or COM Interoperability related mechanism which is not performance-effective usually, very difficult to program and maintain. C++\CLI provides features needed to develop managed wrapper over the legacy C++ RTI library by using C++ like syntax. There are many features of this new programming language which can be found in [58], [68], [71].

4.3 Assemblies

As mentioned in [72] every modern execution environment has a notion of "Software Component", which is called assembly for this .NET based solution. In CLR, an assembly can be an executable (".EXE") or a library (".DLL"). The use of .NET assembly is similar to managed code in such a way that the code in an assembly is first compiled into CIL that makes up the assembly, and then

compiled into machine language at runtime by CLR. For our study, we use it as a Dynamic Link Library (DLL) which is in fact a CIL code with a manifest file that contains information about itself and its content. Every assembly should have a file that contains manifest which is a set of tables containing metadata that lists the names of all files in the assembly, references to external assemblies, and information such as name and version that identify the assembly [64].

The assembly in .NET plays three important roles. The first one is code deployment. Independent from the kind of application, e.g. a stand-alone program, a user interface control, or a DLL library as in our case, all CIL code is packaged into an assembly. Second role is version control. The version information is another field that manifest file contain in itself. The third role is security. The access to members and types that exist in assembly can be set through modifiers provided by programming.

.NET Assemblies are selected by means of easy deployment (e.g. simple copy/remove commands can be used to transfer/delete them), self-informative structure (e.g. containing version info in itself) and, most importantly for our purposes the capabilities provided for dynamic loading of assemblies at run-time.

CHAPTER 5

MODEL BASED SOLUTION TO THE FOM INDEPENDENCE PROBLEM

In this chapter a model-based solution to the FOM Independence problem (FIP) is introduced in detail. The metamodel proposed in [10] is utilized as the basis. HOMM is extended with the addition of new model elements, constraints and an additional Interpreter which is described in Appendix-C. While doing this, model based approach is used as in HOMM and applies it to FIP with employing .NET 2.0 technologies as well. A .NET 2.0 compatible `RTIDotNet1516` library, which is described in Appendix-D in detail, is developed to make developing .NET compliant applications possible. Most importantly, the Conversion Component that is responsible from performing necessary conversion operations is automatically generated after the developer enters the custom conversion code for the correspondence maps.

In this chapter, the model based solution to FOM Independence problem is expounded with detailed descriptions. First, the details of FOM Independence Problem is described with some common terminology that is used in this study are described. After introducing the problem definition, the three-phased solution is proposed with the description of example that illustrates the solution method. In modeling phase, FOM Independence Metamodel (FIM) is explained along with related modeling activities, then automatic code generation phase is explained with model analysis and source code generation steps, and finally component generation phase is described. Fourth section describes the example which is also used to illustrate solution steps in detail with alternative usage scenarios.

5.1 Problem Definition

In this section, the FIP is analyzed and some supporting definitions like Dependency Levels are given in details.

The FIP is a common problem in HLA community. It becomes more pronounced as HLA adoption is increased and more HLA applications are developed. As described before, the FIP is encountered when a federate which is developed for previously designed federation is required to be reused in newly developed federation. Here, the mentioned reuse does not involve change in federate code. In fact, by changing the federate application source code it can readily join into new federation. However, this federate now can not join to previous federation because of changes. Nevertheless change of federate code may not be possible or easy for complex legacy federates.

A note on terminology: In HLA standard, the terms federate and federate applications have different meaning. Yet for the sake of brevity we use “federate code” instead of “federate application code” as no confusion can arise.

The most obvious reason for this problem to occur is that the business domain and the concepts represented with these federations or federates might change from one federation to another.

The most important result of this problem is necessity to change federate code and model according to new federation and rebuilt the federate application. This might not be an effort for small federates and federates that are not being probably participated in more than two federations, but it could be really a big problem for large and dynamic federates which possibly be used with more than one federation. There could also be some cases in which federate could easily join into new federation just by changing some configuration files or FDD file where no recompilation or built of federate code is needed.

The FIP appears between two federations, which are designated as the Source Federation and Target Federation. The Source Federation is the federation which

contains the Source Federate, which is required to join into Target Federation as Target Federate. Usually the Source Federation and Source Federates are developed before and desired to be reused with the newly developed Target Federation. The new role that Source Federate performs in this Target Federation is stated as Target Federate. This relationship is illustrated in Figure 5-1.

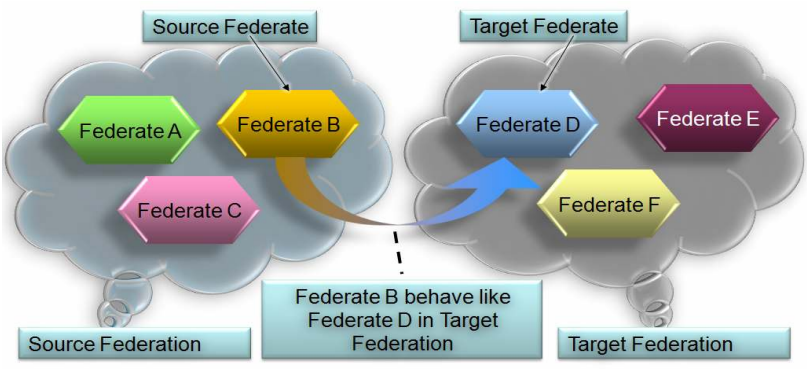

Figure 5-1 Target and Source Federation/Federate relationship

The FOMs of these two federations are different, which are called the Source FOM and Target FOM respectively. To be able to reuse Source Federate as Target Federate in Target Federation, the developer needs to define the relationship between the `Attributes/Parameters` of Source and Target federates that are defined in their FOMs.

To draw the boundaries of the problem, we define the problem in three dependency levels where increase in level of dependency with complexity and difficulty causes more effort, time and cost. The Figure 5-2 depicts these three layers.

Figure 5-2 FIP Dependency Levels

- **Level 1. Syntactic Differences**

Syntactic Differences is the first level of dependency. This level contains the dependency which is resulting from differences in names of HLA Object/ `Interaction Classes`, `Attributes` and `Parameters` which are semantically same and in fact represented with same data structures and types for in both Source and Target Federations. This kind of differences can easily be solved initially by developing federate code independent from HLA Object/Interaction Class, Attribute and Parameter names through reading FDD or similar file if the federate make uses of these string values. Consequently, when a federate attempts to join into a new federation where only these names change, the source code does not needed to be recompiled or built, only the FDD or configuration files used needed to be changed. For instance, assume that, there is a Source Federate in which the temperature Attribute is named as "Temperature" and defined as floating number. Now this federate wants to join to another federation in which temperature attribute is named as "AverageTemperature", but it is still defined as floating number. This federate can not be used directly with Target Federation and federate code needs to be changed and rebuilt, if this "Temperature" Attribute name is hard coded and used in their code. But obviously this can easily be avoided by employing some programming tricks that employ mentioned configuration files or FDD files like reading attribute names before using them with HLA services.

- **Level 2. Syntactic and Representation Differences**

The second level of dependency is caused by the Syntactic and Representational Differences. These differences come out when the same concept that is used in

one federation represented with different data types, names or representations (units) without changing its semantics in other federation. The most obvious difference is caused from the use of different data types for `Attributes` or `Parameters`. For instance, in our Source Federation, "Temperature" is represented as Celsius and defined as floating point. In Target Federation it is still defined as Celsius but it is defined as integer. For floating point and integer case, federate code change may not be necessary, but if it is defined as user defined type or complex data type, the code change might be needed. The difference at this level might also be resulted from differences in representation. For instance, in Source Federation "Temperature" is defined as integer and represented as Celsius and in Target Federation it is defined as user defined Real Number type and represented as Fahrenheit. In this case, at least a conversion between Celsius and Fahrenheit is needed in addition to data type conversions between integer and Real Number type.

- **Level 3. Semantic & Syntactic Differences**

The third and most problematic level of dependency is resulting from Semantic & Syntactic Differences These differences are mainly resulted from changes in semantics, and parallel to this, the syntactic changes happen. At this level of dependency, although the concepts used in different federations are related, the semantics, representation and consequently syntax might change drastically from Source to Target Federation. For instance, "Location" is represented as Military Grid Reference System (MGRS) in Source Federation and in Target Federation it is represented as floating number Latitude and Longitude. Both of these two coordinate systems used for showing and representing locations; however their semantics are very different from each other. In this case, federate code needed to be changed to both handle new coordinate system and data types which might also affect the internals of federates that depend on this coordinates system. Nevertheless, the conversion among different kinds of coordinate systems might need third party libraries to be involved in federate code to convert coordinate from one to another. As it can be seen, all these cause chain affect on federate code which make it necessary to change federate code.

This study proposes a model based solution for FIP which might be occurred at different levels as shown above. In addition to these dependency levels, developer can always construct mappings among irrelevant Source and Target Federation `Attributes` and feed necessary data to Target Federation or use incoming data from Federation through `Conversion Method` mechanism. This approach can be useful for federation that required to be fed with constant data.

5.2 Three-Phased Solution

The solution proposed for FIP employ Model Based approach mentioned in chapter 3 and .NET 2.0 technologies in chapter 4. The scope of this study contains four groups of federation services, Federation, Declaration, Object Management and some of Supporting services. Moreover some of services provided more than one service alternatives like service "UpdateAttributeValues" of Object Management that uses Time and Data Distribution Managements are also excluded. The complete list of implemented services can be found in Appendix B.

To solve the problem we decided to define logically separated group of steps, phases, to make process easy to follow, so the solution is grouped into three sequential phases which are modeling, automatic code and component generation phases. The modeling phase contains activities like Modeling of Federation and construction of `Correspondence Model`, then automatic code related with this and federation related model is generated by developed `Interpreter` software and finally the Conversion Component that will be used in Federation Execution is generated by taking the developer-filled `Conversion Method` templates.

In this section, these three phases are described in detail with accompanied activities. In section 5.3.1, in addition to modeling phase activities, the FIM, correspondence model verification and some constraints are explained. In section 5.3.2, the automatic code generation phase is detailed through model analyze and source code generation sub-phases in which generated source codes and their

usage is described briefly. In section 5.3.3, the usage of Conversion Methods and generation of Conversion Component is explained.

The three-phased solution is illustrated in Figure 5-3. In this figure, important activities that performed for each phases, inputs and outputs are illustrated. The upper part represents modeling phase, bottom-right circle represents automatic code generation phase and bottom-left circle represents component generation phase. The phases that need user involvement are shown with person icon, the rest is performed automatically.

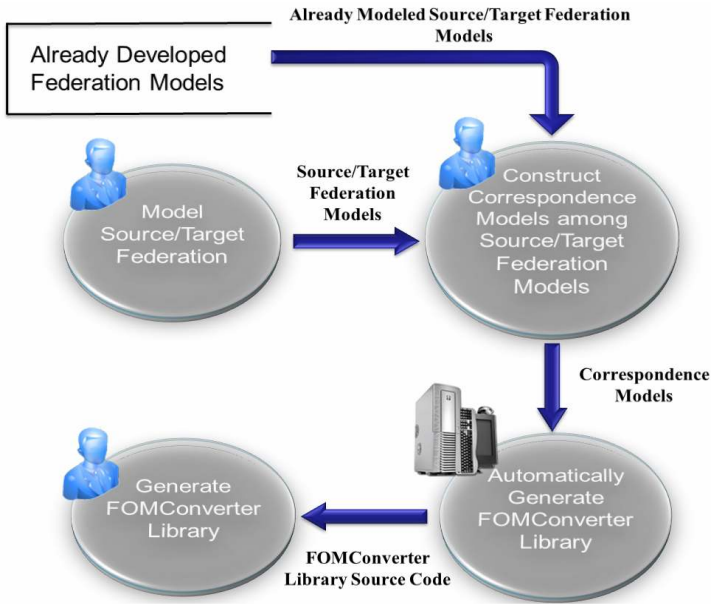

Figure 5-3 The three phases of proposed solution and their relationship

The Figure 5-4 shows overview of software components used in this study and their relationship with each other. In a typical HLA application only Native RTI and federate code is used. The other parts represent software components developed for this study. Whenever federation change is needed, only Conversion Component is changed no source code is changed. The darker numbers represent the path that federate initiated calls follow and the other numbers represent the path that RTI initiated callbacks follow.

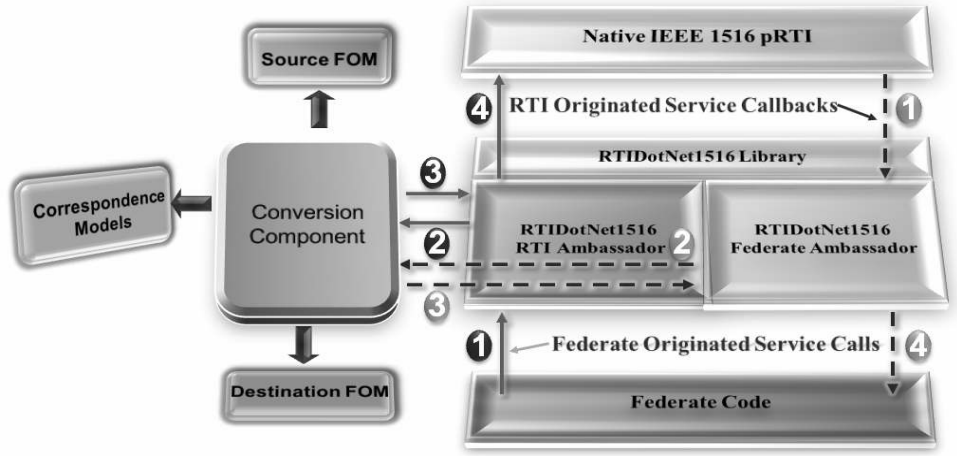

Figure 5-4 Overview of software components used in this study

Before starting to next section, example used to describe technical solution is described here. The example contains two federations; War Game, the source federation, and Command, Control and Communication (C3) Federation, the target federation. The War Game federation which contains Aircraft, Air Defense System and Environment federates is Source Federation and the C3 federation which contains Radar, Command Control, Target and Meteorology federates is Target Federation. The federation developer wants to use her Source, Environment and Aircraft Federates as Meteorology and Target Federate in C3 federation. The solution that solves FIP at different levels is explored in succeeding sections. Figure 5-5 shows the high level War Game federation design model.

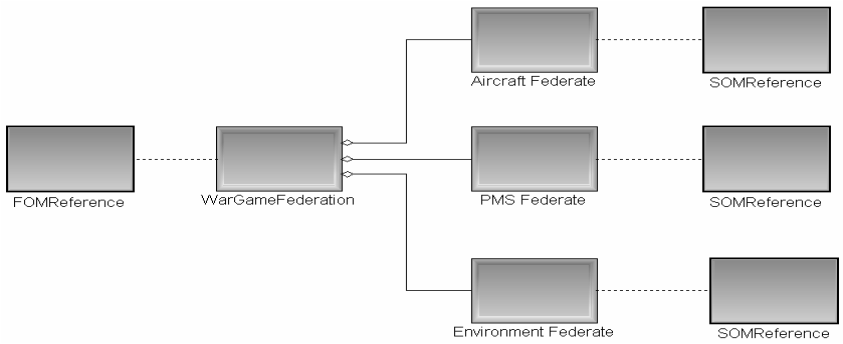

Figure 5-5 The War Game Federation Design top level model

5.2.1 Modeling Phase

The modeling phase covers Modeling of Federations, Correspondence Models and verification of Correspondence Models. The Correspondence Model is simply model representation of mapping from Source FOM Attributes/Parameters to Target FOM Attributes/Parameters which are modeled according to FIM. Theses mappings are created among two or more Attributes/Parameters according to semantics of Attributes/Parameters. The collections of all these Correspondence Models make up Adaptation Model. One Adaptation Model is created for each new federation.

In this thesis, nothing is done related with Modeling of Federations which is already made in HOMM, but new constructs are added. As a result of using HOMM, developer can also model her Federations through the modeling environment provided by GME that uses FIM. Moreover, developer can bring and import her already modeled federations into FOM modeling environment and start to construct Correspondence Models. She can also model either Source or Target Federations or both of them from scratch and then construct Correspondence Models. The steps necessary to import HOMM compatible models are described in Appendix A.

The studies related with FIP are either providing a table or list based mapping, which is not very intuitive and flexible. A significant attention is paid on the mechanism that is used for building relationship among Attributes /Parameters to make is easy but also simple to describe. We provide a simple mapping mechanism to create Correspondence Model through our FIM based modeling environment. Different from previous approaches, the mappings are models and they called Correspondence Model in our approach. This is not only easy, but also very intuitive to use in which you just decide on attributes and then by using mouse create a relation/mapping between them. The most important factor lies behind this easiness and flexibility is the modeling

environment which is based on FIM. The next section explains FIM through additions that we have made to HOMM and then mechanism employed to verify `Correspondence Model` and constraints that defined in FIM are described.

5.2.1.1 FOM Independence Metamodel (FIM)

As mentioned before, FIM is based on HOMM which is described in section 3.6. To be able to construct `Correspondence Model` and make use of HOMM, some additions have been made to original HOMM as described below, these additions are made to OMT Core folder and Object Model paradigm.

- **AttributeConverter Atom**

The `AttributeConverter` Atom (for definition of Atom section 3.5 can be consulted) is defined under HOMM's Classes paradigm sheet of OMT Core Elements. It is used to construct correspondence among `Attributes` owned by `Object Classes` defined in FOM. To build correspondence, developer decides on related `Attributes` for Source and Target FOM which will be inputs and outputs to `AttributeConverter` respectively. The each input to this `AttributeConverter` comes from `Attributes` of Source FOM and they can be from one or more `Object Classes`. The each output is provided to `Attributes` of Target FOM and they can also be from one or more `Object Classes`. Each `AttributeConverter` need at least one input and one output from `Attributes`. The metamodel portion that illustrates this atom and its relationship with other metamodel elements is shown in Figure 5-6.

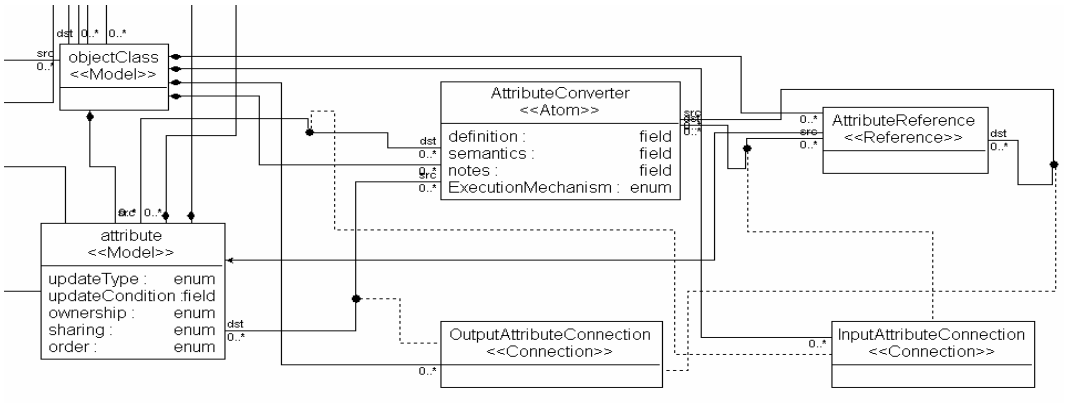

Figure 5-6 `AttributeConverter` Atom & other related metamodel elements

AttributeConverter Atom Attributes:

- **Execution Mechanism:** The execution mechanism of `Attribute-Converter` can be one of Publish/Subscribe Directed or Bi-directional mechanisms. According to this choice, the `Conversion Method` templates are generated in next phase. `Conversion Method` is used to define the conversion logic among the mappings constructed above. The number of templates is determined at modeling phase according to chosen Execution Mechanism.

When Publish Directed is chosen, the RTI Ambassador Services that are called with the attributes come from Source Federates are interfered and these attributes are fed into developer defined `Conversion Methods`, processed at runtime and then sent through RTI Ambassador services to interested federates automatically. The Target Federation originated calls that goes to source federate through Federate Ambassador are not fed into `Conversion Methods` and also not provided to source federate to prevent any unexpected behavior. For this option only one `Conversion Method` Template, "[AttributeConverter-Name]PublishDirectedFunc" is generated.

The Subscribe Directed mechanism is similar but applied for opposite direction. In other words, Target Federate originated calls that comes to Source Federate through Federate Ambassador callbacks are interfered and these are fed into

developer defined `Conversion Methods`, processed at runtime and then converted attributes are provided to Source Federate through Federate Ambassador callback functions automatically. Similar to previous option, one `Conversion Method` Template is generated, "[AttributeConverterName]-SubscribeDirectedFunc".

The Bidirectional Mechanism interferes with all services without considering where it originated from and fed `Attributes` into related developer defined `Conversion Methods` and then passes converted values. Although the Bidirectional Mechanism is ideal for FIP, the other options are also added for sake of completeness. The both of `Conversion Method` Templates are generated for this option.

- **Definition**: This attribute contain information about this Atom.
- **Semantics**: Semantics for `AttributeConverter` can be given.
- **Notes**: Additional notes related with `AttributeConverter`.

The Figure 5-7 shows an example usage of `AttributeConverter` in which a mapping between two `Attributes` that represent temperature from Environment and Meteorology Federates are constructed.

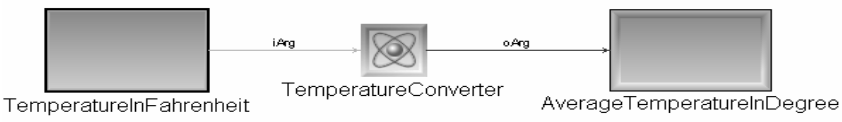

Figure 5-7 An example usage of `AttributeConverter`

○ **AttributeReference Reference**

The AttributeReference Reference (for definition of Reference section 3.5 can be consulted) is defined under HOMM's Classes paradigm sheet of OMT Core Elements. This item is created to make it possible to use an Attribute in other models than it originally defined. Usually, `Attributes` that will be used for mapping are stay in different places and they need to be brought together to relate

them with each other. An AttributeReference can be used and behave as if it is original Attribute. The only thing that is needed to use AttributeReference is dragging the original Attribute item onto AttributeReference. The developer can go to the model where original Attribute stay by clicking on corresponding AttributeReference. Figure 5-7 shows an example usage of AttributeReference where it is used to refer Temperature Attribute as TemperatureInFahrenheit.

AttributeReference can be named differently from the Attribute it refers, the default name is set as AttributeReference and its default shape can be seen in Figure 5-7. The relationship of AttributeReference with `Attribute-Converter` is illustrated in Figure 5-6.

o **InputAttributeConnection & OutputAttributeConnection Connections**

The InputAttributeConnection & OutputAttributeConnection Connections are defined under HOMM's Classes paradigm sheet of OMT Core Elements. These items are created to connect `AttributeConverter` with `Attributes` and AttributeReferences. These are defined separately to make inputs and outputs of `AttributeConverter` distinct and also the names of arguments of `Conversion Methods` are generated according to names of these connections. These two types of connections are shown in different colors and direction of arrows; InputAttributeConnections which connect source `Attributes` or AttributeReferences to `AttributeConverter` are shown in green and OutputAttributeConnections which connect `AttributeConverter` to Target `Attributes` or AttributeReferences are shown in Red. The Figure 5-7 shows these two kinds of connections where arguments names of generated method templates will be "iArg" and "oArg".

While creating connections, the kind of connection that developer made is determined according to order of item selection. For instance, if developer first clicks on an Attribute and then clicks on `AttributeConverter` it is determined as InputAttributeConnection and OutputAttributeConnection if she selects in reverse order.

The possible connection combinations are defined in FIM as shown in Figure 5-6 which only permits to use these connections among AttributeConverters, `Attributes` and AttributeReferences. An attempt to connect an `Attribute-Converter` with itself or any inappropriate combination is prohibited by constraints defined.

o **ParameterConverter Atom**

The `ParameterConverter` Atom is defined under HOMM's Classes paradigm sheet of OMT Core Elements. It is used like `AttributeConverter`, but defined to construct correspondence among `Parameters` owned by `Interaction Classes`. All attributes of `AttributeConverter` are also created for `ParameterConverter`. The metamodel portion that illustrates this atom and its relationship with other metamodel elements is shown in Figure 5-8.

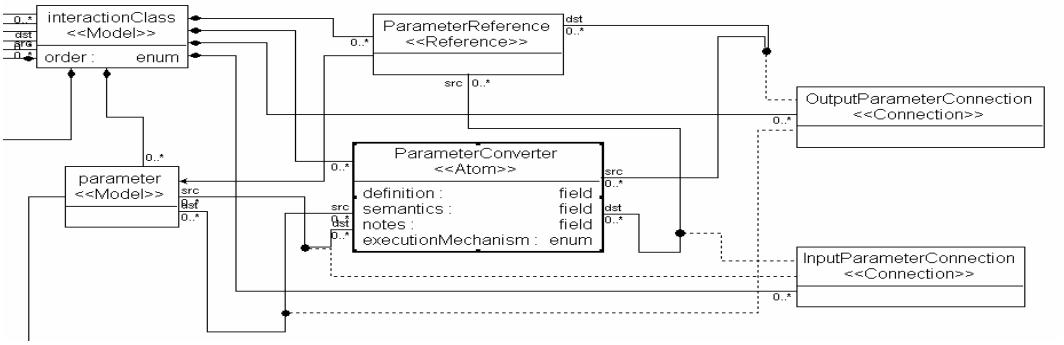

Figure 5-8 `ParameterConverter` Atom&other related metamodel elements

o **ParameterReference Reference**

The ParameterReference Reference is defined under HOMM's Classes paradigm sheet of OMT Core Elements. It is created for same purpose with AttributeReference, but specifically for `Parameters`.

o **InputParameterConnection & OutputParameterConnection Connections**

The InputParameterConnection & OutputParameterConnection Connections are defined under HOMM's Classes paradigm sheet of OMT Core Elements. These items are created for same purpose with InputAttributeConnection and OutputAttributeConnection, but specifically for `Parameters`.

o **SynchronizationPointMapper Atom**

The `SynchronizationPointMapper` Atom is defined under HOMM's Synchronizations paradigm sheet of OMT Core Elements. The Synchronization Points are stated as strings and provided to RTI. The `Synchronization-PointMapper` accepts Synchronization Points from Source and Target Federation as input and output respectively. Its usage is similar to `AttributeConverter` and `ParameterConverter`, but `Synch-ronizationPointMapper` do not accept more than one incoming or outgoing connections. The Figure 5-9 shows related paradigm sheet with `SynchronizationPointMapper` and its relationship with other elements;

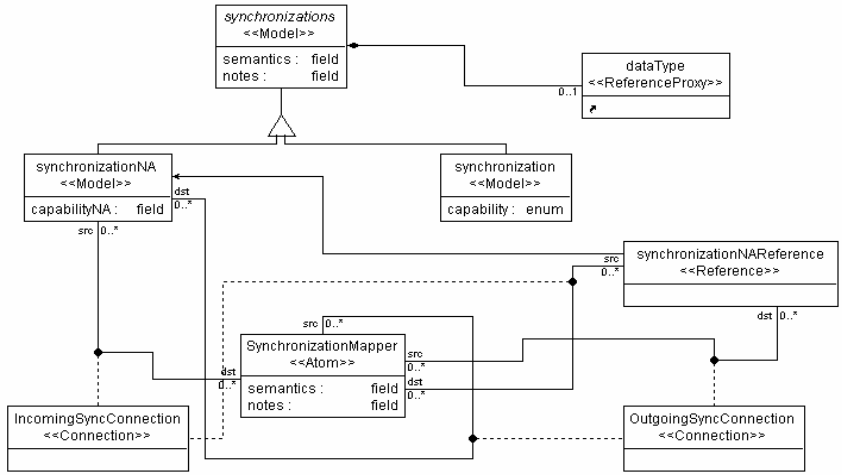

Figure 5-9 Synchronizations paradigm sheet

SynchronizationMapper Atom Attributes:

- **Semantics**: Semantics for SynchronizationMapper if needed can be given.
- **Notes**: Additional notes related with SynchronizationMapper.

The Figure 5-10 shows an example usage of `SynchronizationPoint-Mapper` in which an example mapping between two Synchronization Points that represent start event are constructed.

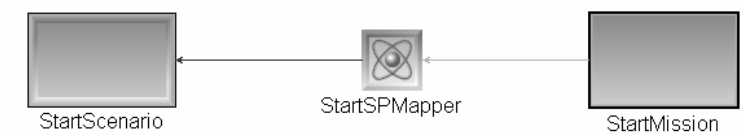

Figure 5-10 An example usage of `SynchronizationPointMapper`

o **IncomingSyncConnection & OutgoingSyncConnection Connections**

The IncomingSyncConnection & OutgoingSyncConnection Connections (are defined under HOMM's Synchronizations paradigm sheet of OMT Core Elements. These items are created to connect `Synchronization-PointMapper` with Synchronization, SynchronizationNA and Synchronization-NAReference. These two types of connections are shown in different colors and direction of arrows; IncomingSyncConnection which connect source Synchronization to `SynchronizationPointMapper` are shown in green and OutgoingSyncConnection which connect `SynchronizationPointMapper` to destination are shown in Red.

The possible connection combinations are defined in FIM as illustrated in Figure 5-9 which only permits to use these connections among `Synchronization-PointMapper` and Synchronization or SynchronizationNA or Synchronization-NAReference.

5.2.1.2 Mapping Formations

The one important issue encountered in modeling phase is how Attributes and Parameters can be mapped during modeling phase and performs necessary conversions during runtime according to this Correspondence Model. When possible scenarios are analyzed according to no of different Object Classes/Interactions that owns Attributes/Parameters involved in mapping, four different formations are realized. These are one-to-one, many-to-one, one-to-many and many-to-many mappings. The factor that used to determine the corresponding mapping formation is the number of the different Object Classes and Interactions that provide Attributes/Parameters to AttributeConverter/ParameterConverter. The Interaction Classes are usually used like events in HLA Federations so the mappings other than one-to-one are not common like Object Classes so the other mapping formations are not given in detail here.

- **One-to-one Mapping**

The one-to-one mapping simply processes the one Object/Interaction Class owned one or more Attributes/Parameters that come from Source Federate and generate similar RTI calls to Target Federation with corresponding Attributes/Parameters and vice versa for calls originated from Target Federation to Source Federate. At first glance, it may seem that all possible FIPs can be solved by using this one-to-one mapping mechanism but there also exist many other situations where data representation or concepts are distributed among many Object Classes in one federation and differently in other federations which make it necessary to handle the other mapping cases. The mappings shown in Figure 5-7 is an example of one-to-one formation.

- **One-to-many and Many-to-one Mapping**

In one-to-many case the converters process the one Object Class owned one or more Attributes that come from Source Federate and generate multiple RTI Ambassador service calls for each different Object Class with

corresponding `Attributes` in Target Federation. When necessary services are executed in Target Federation which is usually more than one, there will be only one Federate Ambassador callback for `Object Class` owned `Attributes` in Source Federate.

Many-to-one mapping is similar to one-to-many case, but in this case Source Federate makes multiple RTI Ambassador service calls and only one Federate Ambassador callback is generated for Target Federation.

- **Many-to-many Mapping**

The most complicated case is many-to-many mapping. In this case, all source federation originated RTI Ambassador calls (e.g. Publish, Subscribe, Update, Send, etc) should be handled according to `Correspondence Model` for each Attribute. The Target Federation originated Federate Ambassador calls (e.g. Discovery, Receive, Reflect, etc should be handled similarly. The all information necessary for this RTI Ambassador and Federate Ambassador service calls are obtained from the models constructed. What and how all these generated are explained in the next title.

Many-to-many mapping case, in fact, contains the other mapping mechanisms in itself, so in following lines, how HLA services handled according to many-to-many mapping case is described by using War Game and C3 federations with CoordinateConverter mapping that given in Figure 5-11.

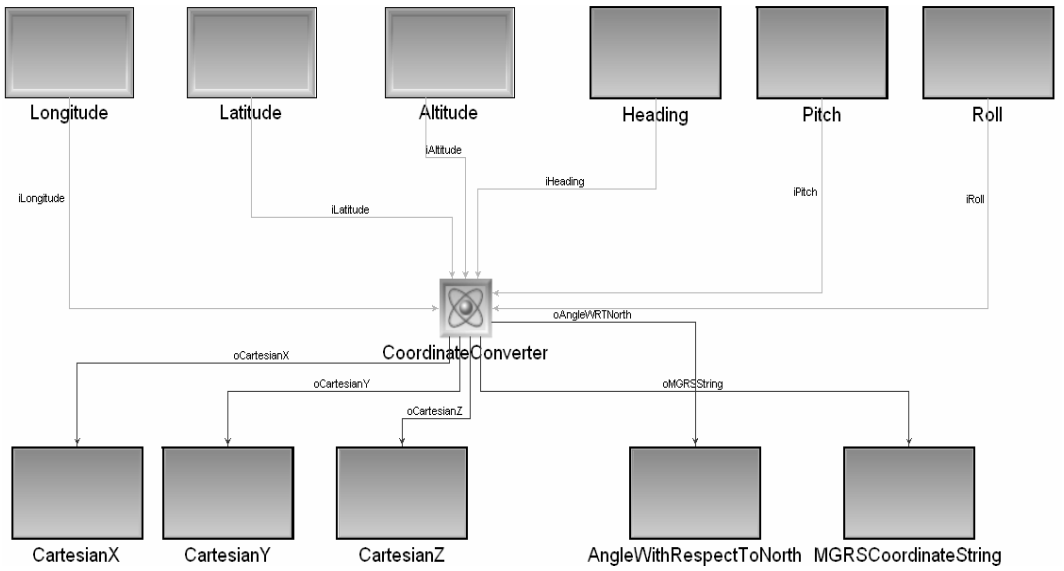

Figure 5-11 CoordinateConverter mapping

First group of services which belongs to Object and Declaration Management are the most important and frequently used services that need to be handled for FIP, so they are given in detail. The rest of the interfered services are described briefly under the service groups they belong.

[Publish|Unpublish]ObjectClassAttributes & [Subscribe|Unsubscribe]Object-ClassAttributes Services:

These four services are belonging to Declaration Management and their corresponding service numbers are 5.2, 5.3, 5.6 and 5.7. In many-to-many formation, developer map more than one `Object Class` owned `Attributes` from Source Federate, so it is assumed that the code necessary to publish and subscribe all these attributes already exist in source federate code. In other words, the `Attributes` provided by Source Federate to Target Federation are being published or subscribed, the order of these service calls is not important.

When necessary publish or subscribe services to all these `Attributes` that owned by Source Federate are called then necessary publish and subscribe RTI Ambassador calls for corresponding `Attributes` as Target Federate will be

made automatically. As stated above, the calls for Target Federation will not be called till all Source Federate `Attributes` subscribed or published.

The case for unpublish and unsubscribe services is different. If a Source Federate calls unpublish or unsubscribe services for any one of `Attributes` it published or subscribed, than unpublish or unsubscribe services will be called for all corresponding Target Federate `Attributes`. Publish or subscribe calls for Target Federation will not be generated till all Source Federate `Attributes` are published or subscribed as stated above. This is done on purpose; because it is assumed that the user-defined `Conversion Method` will need all these `Attributes` to perform necessary conversions for publish directed execution mechanism.

The information like which Source Federate owned `Attributes` are used as input for a `AttributeConverter`, their `Object Classes` and all other necessary data for all these services executions and control are automatically generated before run-time in automatic code generation phase.

A probable scenario of PublishObjectClassAttributes/UnpublishObjectClass-`Attributes` for CoordinateConverter is given in Figure 5-12.

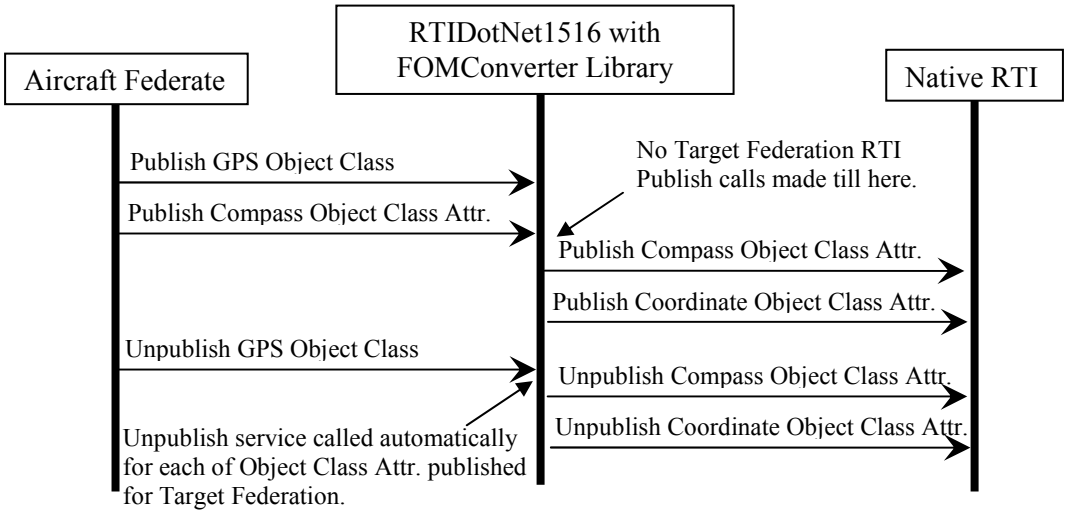

Figure 5-12 Publish/Unpublish services scenario with many-to-many formation

The scenario for Subscribe and Unsubscribe services are similar.

RegisterObjectInstance & DiscoverObjectInstance Services:

These two services are belonging to Object Management and their corresponding service numbers are 6.4 and 6.5. The register RTI Ambassador service is called by Source Federate to register `Object Class Instances` of previously published `Object Classes` and other one is initiated by Federate Ambassador to inform Source Federate that a new `Object Class Instance` of previously subscribed `Object Class` is discovered.

For one-to-one formation case, when a Source Federate registers an `Object Class Instance` the same call for corresponding instance in Target Federation is called automatically. For many-to-many case, we follow similar approach as we did in Publish/Subscribe services where we wait for Source Federate to register `Object Instances` for all `Object Classes` it published and then make necessary register Object Instance calls to Target Federation for each set of `Object Classes` in it. Assume that we have two `Object Classes` that provide `Attributes` as input to `Attribute-Converter` and three `Object Classes` of whom `Attributes` used by Target Federation. Whenever Source Federate registers two `Object Instances`, three `Object Instances` will be registered for Target Federation automatically.

Similar approach is pursued for DiscoverObjectInstance in such a way that the wait will be done for Federate Ambassador initiated DiscoverObject callbacks till Target Federation register all `Object Instances` that Source Federate subscribed before and then Source Federate will be informed through Federate Ambassador callbacks. A probable scenario for these two services is shown in Figure 5-13 using CoordinateConverter mapping.

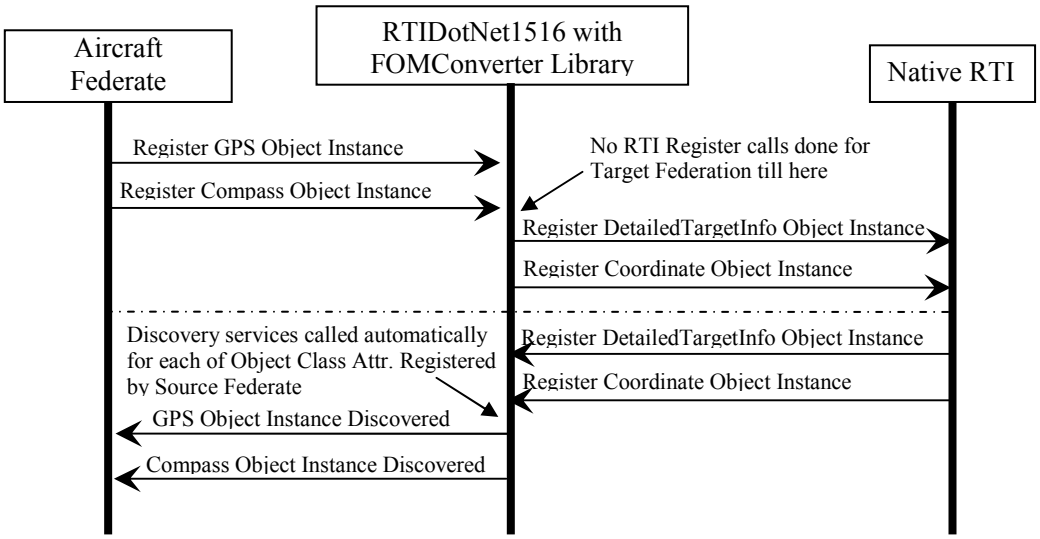

Figure 5-13 Register/Discover services scenario with many-to-many formation

UpdateAttributeValues & ReflectAttributeValues Services:

These two services are belonging to Object Management and their corresponding service numbers are 6.6 and 6.7. These are the key services in FIP that commonly used in HLA applications. These are also the services that make use of user-defined `Conversion Methods`. These services are called and applied for each `Object Instances` that are registered or subscribed as mentioned above. These calls will be discarded if all necessary `Object Instances` are not registered by Source or Target Federate to prevent any anomalies.

The application of `Conversion Methods` to one-to-one formation is straight-forward for these services in such a way that when an update or reflect service is initiated, the corresponding `Conversion Method` will be called for given instance. The situation is a little bit different for many-to-many formations. The most crucial point of using these services with many-to-many mapping is the time when user-defined `Conversion Methods` will be called, i.e., should we call conversion method for each new set of Object Instance Attribute values or should we call `Conversion Methods` whenever one of Attribute value is updated? In this study we employ second approach by allowing this update after all related

Attributes updated their values once, and then the Conversion Methods are triggered with each UpdateAttributeValues call and corresponding service calls done automatically with each of these calls. We choose this approach, because HLA permit us to use these two services with only one Object Instance's attributes update for each call (i.e. you cannot update two object instance attributes in one call, you need to make two distinct calls) and also the first approach might cause inconsistencies and long waits if Attributes are not updated at regular periods.

Another approach considered but not chosen is use of counters for each Attribute in such a way that counters are increased with each update and the Conversion Method will be triggered when certain counter values are reached. Although these counter values might be determined according to update period of each corresponding Attribute, it can be very difficult to find correct values and combination for each Attributes owned by Object Classes. Each Object Instance and corresponding Attribute Values is kept and Conversion Methods are called with related Object Instance's values.

ReflectAttributeValues service call works similarly, just in the opposite direction and triggered with Federate Ambassador ReflectAttributeValues.

A probable scenario of UpdateAttributeValues service for CoordinateConverter is shown in Figure 5-14.

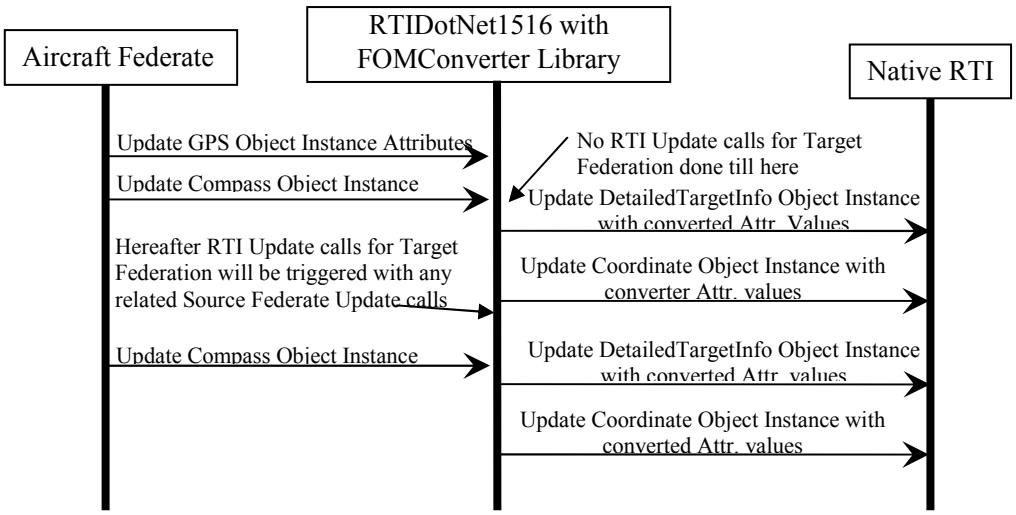

Figure 5-14 Update service scenario with many-to-many formation

SendInteraction & ReceiveInteraction Services:

These two services are belonging to Object Management and their corresponding service numbers are 6.8 and 6.9. These are other two important services that make use of user-defined `Conversion Methods`. The `Conversion Methods` are called in a similar way to update and reflect services, but different from `Object Classes`, `Interactions` do not have an instance so when these `Conversion Methods` should be called is not a problem. The necessary `Conversion Methods` will be called when developer publishes or subscribes to all necessary `Parameters` like in `Attributes`.

Federation Management Services:

This group of services is not affected from the kind of mapping formations, but there are still issues that need to be interfered. There are two kinds of interferences with these RTI Ambassador calls.

The first kind of the interference is usually done to string values that represent some arguments like “Federation Execution Name”, “FDD File Path”. When a Source Federate, which uses a different FDD file from Target Federation, initiates a CreateFederationExecution service to create Target Federation, the Federation

Name and FDD file path are automatically provided with additional RTI call. This approach is also used for other similar services like DestroyFederationExecution and JoinFederationExecution. A probable scenario of CreateFederationExecution service for CoordinateConverter is shown in Figure 5-15.

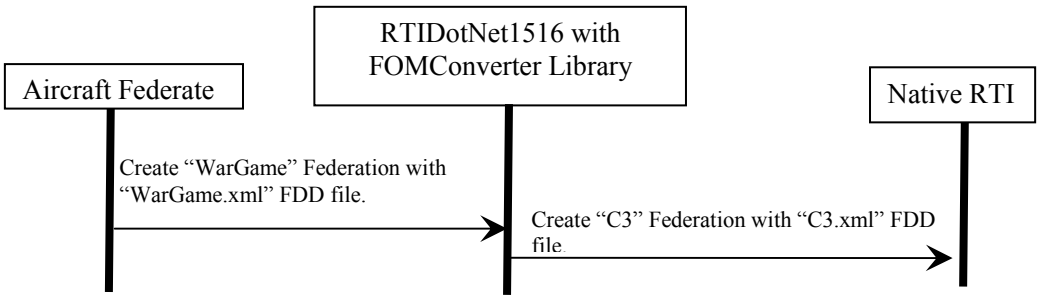

Figure 5-15 A scenario for Create service

The second kind of the interference is done for synchronization point related RTI services like RegisterFederationSynchronizationPoint. After developer defines the mapping points among Source and Target Federation, whenever a service uses this mapping is called, the other member of mapping which defined as string is provided to Source or Target Federate with additional corresponding RTI call.

Object Management Services:

The approach employed for these services belong to this group is explained above in detailed. The approach for other services in this group is similar. For instance, consider ChangeAttributeTransportationType service when developer calls this service all AttributeConverters are checked whether they use any of `Attributes` given with this service call. If they contain, then this service is called for all corresponding `Attributes` belongs to Target Federation. Similarly, when Target Federation initiated a ProvideAttributeValueUpdate service call, all related Source Federates are informed through Federate Ambassador ProvideAttributeValueUpdate callbacks.

RTI Support Services:

As mentioned in section 2.3, only some services of this group are handled in this study like GetObjectClassHandle or GetAttributeName. This kind of services are usually called by federate code to identify or request the information that RTI implementation assigned for `Object Classes` and Instances, `Interactions`, `Attributes` and `Parameters`. To be able to behave like Target Federate in Target Federation, the Source Federate should be supplied with this information which is different from the values hold by RTI because of different FDD files and FOMs. To solve this, `FOMConverter` library behave like RTI and assign unique identifiers to all necessary elements. This assignment is not performed at execution time because of performance considerations, so these are determined at automatic code generation phase and loaded to memory with start of Federation Execution. Whenever Source Federate calls any of services belongs to this group, the predetermined values are returned. These predetermined values are also used to hash Source and corresponding Target Federate `Attributes` and `Parameters`.

5.2.1.3 Model Verification and Constraints

The other important issue in modeling phase is the Model Verification. The most important purpose of this verification process is to prevent developer to construct an inappropriate or potentially problematic `Correspondence Models`. The verification of this model can be performed using the infrastructure provided by GME or it can be verified programmatically when developer complete modeling. In our study, both of these approaches are employed.

While developing the metamodel, GME allows developer to define constraints and relate them with metamodel elements to be used in modeling. The constraints are defined with a OCL like language. There are many default options for defining constraints and events that will trigger the check of these constraints. For instance, developer can define a constraint which will be checked when developer saves her model or immediately after the model element is inserted. For details of constraint creation in GME, [55] can be consulted. An example constraint that we define for

`AttributeConverter` and `ParameterConverter` is "HasAtLeastOneOut goingArgument". This constraint controls whether created AttributeConverters or ParameterConverters have at least one outgoing connection to `Attributes` or `Parameters` when model closed. The equations, constraints definitions, are like;

"*self.attachingConnections(OutputAttributeConnection) -> size >= 1*"
"*self.attachingConnections(OutputParameterConnection) -> size >= 1*"

The other constraints defined for `AttributeConverter` and `Parameter-Converter` are;

- "HasAtLeastOneInputArgument", which make same control for input connections. The equations for this constraints are;

  "*self.attachingConnections(InputAttributeConnection) -> size >= 1"*
  "*self.attachingConnections(InputParameterConnection) -> size >= 1*"

- "UniqueConverterNamecontrols", which controls `Attribute-Converter` and `ParameterConverter` names for uniqueness. The equations for this constraints are;

  "*project.isNameUnique(self.name, AttributeConverter)*"
  "*project.isNameUnique(self.name, ParameterConverter)*"

The Figure 5-16 shows FIM a portion related with these constraints.

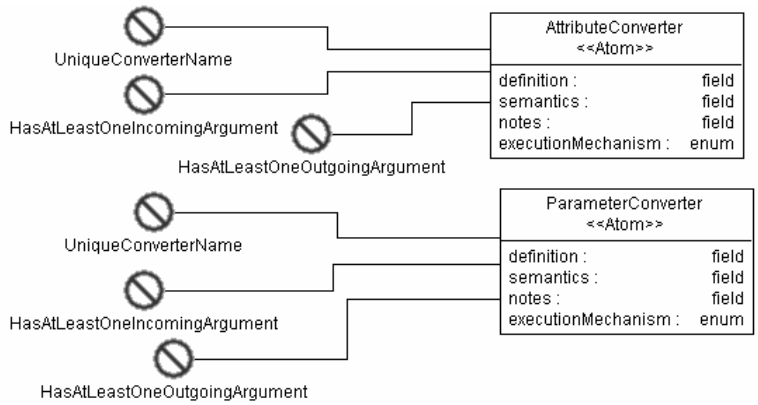

Figure 5-16 Constraints defined for FIM

There is also a verification step in the beginning of automatic code generation phase which is done programmatically by `Interpreter` software. When developer commits her model, it is being checked by walking on models constructed and informs developer about the issues detected. The one verification made at this step is the recursive connections among converters and `Attributes` or `Parameters`. The developer might define both input and output connections among same converter and `Attribute` which is not legal, so developer is warned about this problem. The control of connections that made from one `Attribute` or `Parameter` to multiple converters and connections that made from different converters to same `Attribute` or `Parameter` is also made at this step. The final control is made for uniqueness of connection names. As stated in previous section, the names of connections are used in `Conversion Method` templates as argument names so they should be unique to prevent compilations errors in component generation phase.

5.2.2 Automatic Code Generation Phase

The next phase that comes after modeling phase is automatic code generation phase. This phase is responsible from gathering necessary information from models constructed and automatically generates source codes for conversion components. Automatic code generation and next phase are mostly controlled by model Interpreter software developed and no modeling activity is necessary.

This phase is started after developer complete construction of `Correspondence Model`. Then she executes the program that is responsible from the activities performed in this phase which called `Interpreter` [55]. The `Interpreter` is an add-in software that can be integrated into GME modeling environment through plug-in mechanism provided. It is provided to developer through a graphical user interface like button component, click of which starts the execution of `Interpreter` and consequently this phase. The programmatic and other technical details of this software is given in Appendix C, here its capabilities in the scope of automatic code generation is described.

5.2.2.1 Model Analyze Sub-phase

Code generation from a PSM typically employs transformations that contain rules about these transformations, e.g. produce a class for each item in model with its corresponding name. Some studies like [41] illustrate some of these patterns like Templates + Filtering, Templates + Metamodel, Code Attributes and API Based Generation. Details of these and other patterns and their comparison can be found in [41]. In this study, API Based Generation is employed. In this pattern, an API which is based on a specific metamodel and target language is provided to write applications to be used in generation of source code. In this study, we employ a very similar approach to API-based pattern, but it does not restrict us to a specific target programming language which is the case for API-Based Generation pattern. This approached is used for this study through the API provided by GME. The classes and methods provided by this API are used to gather information from models. Additional code is written to generates source code in a brute force way, i.e. the code is generated using methods and constructs provided by I/O libraries like "fstream.h".

The GME provides three APIs. These are Builder Object Network (BON) version 1.0 and 2.0, and Meta Object Network (MON). In this study, BON2 which defined as a three-layered architecture is used because of its easy and comprehensive programming interface. These three layers are described briefly in Table 5.1.

Table 5.1 Three layers defined by BON2 [55]

Layers	**Layer Description**
Layer 0 (COM Layer)	Programmable interface of GME which is the lowest level.
Layer 1.a (Implementation Layer)	The core of BON2 which contains COM operations.
Layer 1.b (Interface Layer)	This sub-layer contains classes and operations which are exposed to the user.
Layer 2 (Wrapper Layer)	This layer contains the high level wrappers which handles the objects' references.

The BON 2 is based on MON and in fact, it depends tightly to the classes defined in it. The provided BON 2 interface is in C++ programming language.

5.2.2.2 Source Code Generation Sub-phase

The automatically generated code at this phase is based on .NET 2.0 technologies which described in chapter 4. The `Interpreter` generates code in C++\CLI programming language as distributed in more than seven files. The brief descriptions about these files are given below. The detailed descriptions and snapshots can be found in Appendix E.

- **"ConverterLib.h"**

This header file contains data type declarations about `Attributes` and `Parameters`.

- **ConverterLib.cpp**

This file contains the empty body blocks of `Conversion Methods` whose prototypes are generated in "ConverterLib.h" file.

- **ConverterTypeLib.cpp**

This file contains implementation of data type support methods whose declarations are generated in "ConverterLib.h".

- **FOMConverterMgr.cpp**

This file contains the implementation of `FOMConveterMgr` methods like class constructor and internal service processing methods.

- **AttributeConverterIDs.h, ParameterConverterIDs.h**

These files contain identifiers that assigned to each of `AttributeConverter` and `InteractionConverter` created in modeling phase for internal purposes.

- **ConverterLibCommon.h, ConverterLibCommon.cpp**

These files contain supporting function declaration and implementations for developer to convert given byte array to Simple Data Types.

5.2.3 Component Generation Phase

The final phase that comes after automatic code generation phase is component generation phase. This phase is about generation of `Conversion Component` from automatically generated codes in previous phase. This component is used for conversions and handling of RTI services during federation execution. Two important steps exist in this phase, the first one covers filling of `Conversion Method` Templates by developer and the second one is about generation of `Conversion Component`. As stated in section 4.3, .NET 2.0 assemblies are chosen to define libraries in our study. There are two important assemblies, developed and used in our study. The first one is `RTIDotNet1516` library and described in Appendix D and the other one is Conversion Component as `FOMConverter` library which is described in this section.

The one critical point in FIP is how, where and when we should define the relationship among `Attributes` and `Parameters` for Source and Target Federation. The answer for how question is in fact answered in previous sections where we choose to provide developer `Conversion Method` templates to be filled by her in any .NET compatible programming language rather than a table or specific language which provides a great flexibility. These templates are generated automatically in automatic code generation phase and filled in this phase.

These `Conversion Methods` are created and embedded into `Conversion Component` not into RTI or any other files. Such approach not only prevents any unnecessary dependency for federates which do not use these `Conversion Methods` or related capabilities, but also an easy way of deployment through these assemblies to use with similar federates. For instance, if these `Conversion Methods` are placed into RTI, or in our case `RTIDotNet1516`, then whenever a conversion operation changes, this library should be compiled and rebuilt which obviously not only affect all federates uses this library but also the federates that uses this RTI library but not `Conversion Methods`. This also concludes our answer for where question.

The final and most important question is when to define these `Conversion Methods`. We decided to let developer define these methods after all modeling and automatic code generation related activities are finished. It is also possible to provide a mechanism which let developer define conversion method itself in modeling phase. Although it is not a problem for small or straightforward conversion operations, it could be very limiting and not flexible to define more complex conversion operations which might also need to make use of other third party libraries. The other reason to leave this definition operation to end is to minimize and, if possible, prevent any redundant effort. For instance, if it was embedded into modeling phase whenever developer need to change the conversion operation whole model need to be re-analyzed, even no change is made to model, and the code related with model should be regenerated. However, in our case developer can change a conversion method and only thing she need to do is regenerate `Conversion Component` which can also be replaced with previous one without terminating ongoing Federation Execution.

The first step in this phase is the filling or development of `Conversion Method` templates generated in previous phase. These templates are generated in "ConverterLib.cpp" file with their input and output arguments, only the body of function is needed to be filled. The arguments names are given according to connection names given in modeling phase. These methods are generated according to `AttributeConverter`, `ParameterConverter` and other modeling elements connected with these converters. The number of these methods are depends on the attribute value selected provided in these converters which is named as "ExecutionMechanism". The execution mechanism of Attribute-Converter and ParameterConverter can be Publish/Subscribe Directed or Bidirectional mechanisms. If developer selects either one of Publish or Subscribe Directed mechanism only one `Conversion Method` template is generated according to choice. If bidirectional mechanism is selected, two `Conversion Methods` are generated. The developer should fill both of these methods; otherwise unexpected or inappropriate behavior might be observed.

The default programming language that should be used for filling `Conversion Method` is the language chosen for automatic code generation which is C++\CLI for our study, but developer can use other programming languages in three other ways. In first way, developer can define a whole new set of transformations to generate source code with given programming language by modifying `Interpreter` software and then put her conversion code into newly generated `Conversion Method` template. The developer can also develop her conversion logic in other programming language which might be a non-.NET compatible language, then embed this code into a .NET assembly and use it directly in generated code through newly introduced C++\CLI "include" mechanism which let developer to include external assemblies by writing "#include <userDefinedAssembly.dll>" at the beginning of file [12]. In the third way, user can inherit `Conversion Methods` from the classes generated for Conversion Component from any .NET compatible programming language.

While developing the `Conversion Methods`, there are some points that should be taken into consideration. First of all, if `Conversion Method` will be used only for mapping like situations mentioned in first level of dependency, the developer only need to assign incoming argument to outgoing argument for publish directed calls and vice versa for subscribe directed calls. The other issue is about multidimensional arrays. The corresponding size of each dimension of array is not known previously in modeling phase so some methods' source code could not be generated. If such case is encountered in automatic code generation phase, generator put arbitrary identifiers like "PleaseEnterSizeOfArray" to these places which cause errors in compilation. The developer needs to fill these places by hand manually. This is also necessary for GetBytes, SetBytes and GetSize methods for these multidimensional arrays. The developer also should allocate necessary memory for outgoing arguments which can cause errors otherwise during federation execution. The developer is responsible from all exception and error handling for `Conversion Methods`.

As mentioned before, some third party libraries might be needed for `Conversion Methods` and they can easily be used by including them directly into this library by following approaches mentioned above for different programming languages. The developer can also customize these automatically generated files or methods according to her requirements as long as complies with the provided interface.

While filling these `Conversion Methods`, developer can feed any output arguments directly without using input arguments which might be very useful for unrelated attributes that are necessary to be supplied for Target Federation.

The second step in this phase is the generation of `Conversion Component`. The generation of this library is nothing but a build process. The interpreter provide makefiles to developer to built this assembly but developer can also built this Conversion Component by using her tools like Visual Studio .NET 2005 with generated Visual Studio files..

The developer can use Conversion Component with its default name "FOMConverterLibrary.dll" which is recognized by `RTIDotNet1516` library if it is located in same directory. She can also change its name as long as it is given to RTIambassador through configuration arguments as shown in Figure 5.17.

A note on code snapshots: The *italic texts* in code snapshots are filled by developer and the other parts are generated automatically.

```
array<String^>^ args = gcnew array<String^>(4);
args[0] = "crcHost = " + CRCHostAddress->Text;
args[1] = "crcPort = " + CRCHostPort->Text;

/// Define custom converter library name
args[2] = "converterLibrary = UserDefinedConverterLibrary";

/// Define custom converter library path
args[3] = "converterLibraryPath = ./";

/// Create a rti ambassador
mRTIAmbassador=rtiAmbassadorFactory->CreateRTIAmbassador(args);
```

Figure 5-17 Snapshot for Configuration Arguments

Once this library is generated, only thing left to developer is placing this Conversion Component to federate application executables folder. This component can only be used by Source Federates, but necessary precautions are taken by RTIDotNet1516 library and it is discarded even though it is placed into Target Federate's working directory. The developer can make use of these components without stopping federation execution by reloading them and she can also use more than one library within same federate application to be a member of different federations simultaneously. Moreover, one library can be used more than one Source Federates of same federation.

5.3 Case Study

To illustrate the method proposed in this thesis, four federates are considered in two federations and then developed. The federations are War Game which contains Environment and Aircraft federates, and C3 which contains Meteorology and Target federates. The Environment and Aircraft federates are already developed and required to be used in C3 federation, so necessary `Correspondence Models` among War Game and C3 federations should be constructed. The classes and attribute names are chosen in such a way that it make easy to see the related attributes among different federates with similar semantics. Moreover, no additional programming is devoted for the parts that are not related with the study like three-dimensional visualization or unrelated HLA services. All necessary models like SOMs, FOMs and data types used in this example are developed using FIM.

The Target Federation is C3 federation and the Source Federation is War Game federation. The setup of these federates and their relationship with each other is given in Figure 5-18.

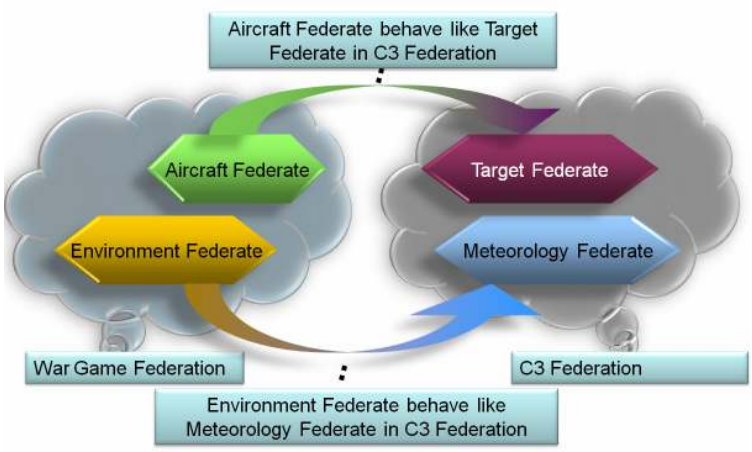

Figure 5-18 War Game Federation, C3 Federation, their members and their relationship with each other

In this section, mostly the related `Object Classes` are given as example which in fact covers all mentioned capabilities in the thesis. Each federate is described separately with owner federation. The related `Correspondence Models` of War Game & C3 `Adaptation Model` constructed for Environment and Meteorology federate are shown in Figure 5-19.

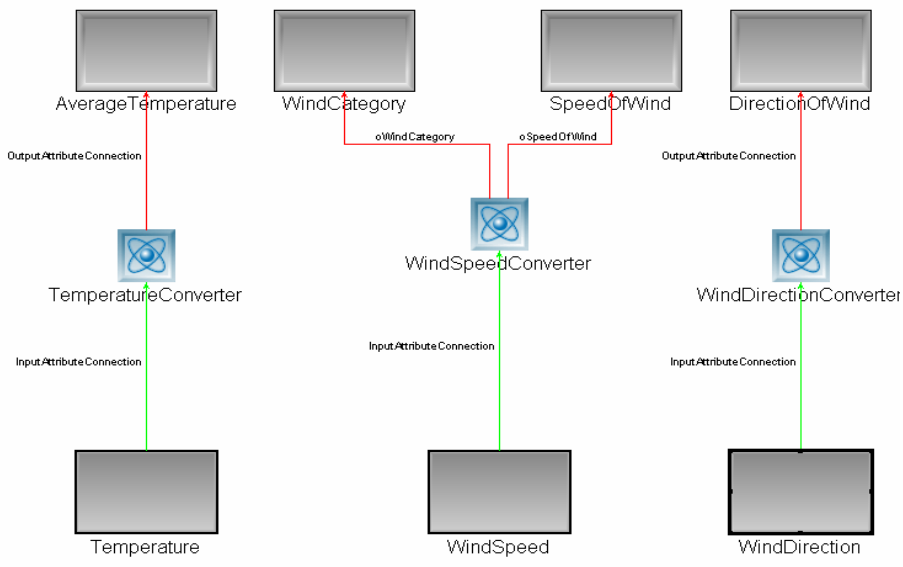

Figure 5-19 `Correspondence Models` constructed for Environment & Meteorology Federates

The filled `Conversion Method` for TemperatureConverter is given in Figure 5-20. This converter converts temperature values to each other which are defined in Celsius for one federate and in Fahrenheit for other federate.

```
/// Conversion method used for publish directed Calls
void FOMConverterMgr::TemperatureConverterPublishDirectedFunc(
      Float% InputAttributeConnection,
      Float% OutputAttributeConnection)
{   /// Write your conversion code here
   OutputAttributeConnection = InputAttributeConnection*1.8f +
                               32 ;
}
/// Conversion method used for subscribe directed Calls
Void FOMConverterMgr::TemperatureConverterSubscribeDirectedFunc(
      Float% OutputAttributeConnection,
      Float% InputAttributeConnection)
{
      /// Write your conversion code here
      InputAttributeConnection = (OutputAttributeConnection - 32)
                               / 1.8f;
}
```

Figure 5-20 `Conversion Method` for TemperatureConverter

The filled `Conversion Method` for WindDirectionConverter is given in Figure 5-21 and Figure 5-22. This converter multiplies or divides Wind Direction values to 10 according to direction of call.

```
/// Conversion method used for publish directed Calls
void FOMConverterMgr::WindDirectionConverterPublishDirectedFunc(
      FloatArray3f % InputAttributeConnection,
      Vec3f% OutputAttributeConnection)
{
      /// Write your conversion code here
      OutputAttributeConnection.X = InputAttributeConnection[0] *
                               10.0f;
      OutputAttributeConnection.Y = InputAttributeConnection[1] *
                               10.0f;
      OutputAttributeConnection.Z = InputAttributeConnection[2] *
                               10.0f;
}
```

Figure 5-21 Publish Directed `Conversion Method` for WindDirectionConverter

```
/// Conversion method used for subscribe directed Calls
void FOMConverterMgr::WindDirectionConverterSubscribeDirectedFunc(
      Vec3f% OutputAttributeConnection,
      FloatArray3f% InputAttributeConnection)
{
      InputAttributeConnection = gcnew array<Float>(3);

      /// Write your conversion code here
      InputAttributeConnection[0] = OutputAttributeConnection.X /
                                    10.0f;
      InputAttributeConnection[1] = OutputAttributeConnection.Y /
                                    10.0f;
      InputAttributeConnection[2] = OutputAttributeConnection.Z /
                                    10.0f;
}
```

Figure 5-22 Subscribe Directed `Conversion Method` for WindDirectionConverter

The filled `Conversion Method` for WindSpeedConverter is given in Figure 5-23. This converter converts speed values to each other which are defined in miles per hour (mph) for one federate and kilometers per hour (km/h) for other federate. There is also a scaled data for Meteorology Federate which is defined as Wind Category. It is calculated according to Wind Speed value updated by Environment Federate and there are four categories which are calm, breeze, storm and hurricane. This category information is not used by Environment Federate, so nothing is done for subscribe directed `Conversion Method`.

```
/// Conversion method used for publish directed Calls
void FOMConverterMgr::WindSpeedConverterPublishDirectedFunc(
      Float% InputAttributeConnection,
      Float% OutputAttributeConnection)
{
      /// Write your conversion code here
      oSpeedOfWind = InputAttributeConnection * 1.609f;

      if( oSpeedOfWind < 100 )      oWindCategory = CALM;
      else if( oSpeedOfWind < 200)  oWindCategory = BREEZE;
      else if( oSpeedOfWind < 300)  oWindCategory = STORM;
      else                          oWindCategory = HURRICANE;
}

/// Conversion method used for subscribe directed Calls
void FOMConverterMgr::WindSpeedConverterSubscribeDirectedFunc(
      Float% OutputAttributeConnection,
      Float% InputAttributeConnection)
{
      /// Write your conversion code here
      InputAttributeConnection = OutputAttributeConnection /
                                 1.609f;
}
```

The related `Correspondence Model` that is constructed for Aircraft and Target federates is shown in Figure 5-24.

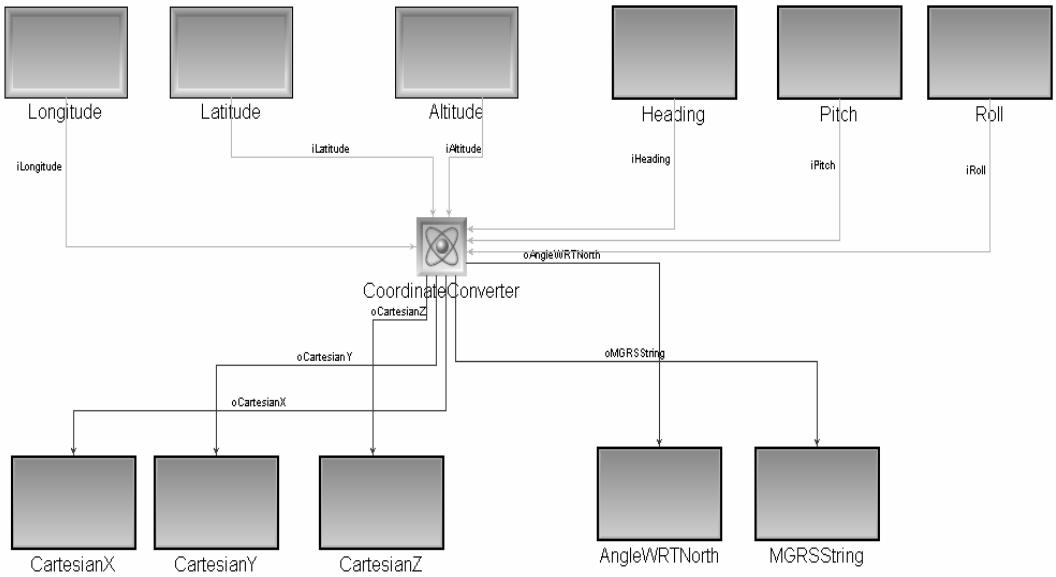

Figure 5-24 `Correspondence Models` constructed for Aircraft & Target Federates

The filled `Conversion Method` for CoordinateConverter is given in Figure 5-25 and Figure 5-26. This converter converts coordinates values to each other which are defined in MGRS, and GeoCentric for one federate and LLDMS for other federate. This `Conversion Methods` shown in Figure 5-25 and Figure 5-26 are also example to use of non .NET 3rd party libraries for this solution. Some of methods that used from this library for conversion logic are "Set_MGRS_Parameters", "Convert_Geodetic_To_Geocentric", etc.

```
/// Conversion method used for publish directed Calls
void FOMConverterMgr::CoordinateConverterPublishDirectedFunc(
            Float% iLatitude, Float% iLongitude, Float% iRoll,
            Float% iHeading, Float% iPitch, Float% iAltitude,
            HLAASCIIstring% oMGRSString, Double% oCartesianZ,
            Double% oCartesianX, Float% oAngleWRTNorth,
            Double% oCartesianY)
{
      /// Write your conversion code here
      char MGRSText[22] = {0};
      double a =  6378137.0;
      double f =  1.0 / 298.257223563;
      char *Ellipsoid_Code = "WGE";

      /// Set MGRS Parameters for conversion
      long status = Set_MGRS_Parameters(a, f, Ellipsoid_Code);
      long Precision           = 5;
      double LatitudeInRadian  = DEGREE_TO_RADIAN( iLatitude );
      double LongitudeInRadian = DEGREE_TO_RADIAN( iLongitude );

      /// Convert given LLDMS coordinates into MGRS
      Convert_Geodetic_To_MGRS(LatitudeInRadian,
                        LongitudeInRadian, Precision, MGRSText);

      /// Pass MGRS string
      oMGRSString = gcnew array<HLAASCIIchar>( 22 );

      /// Copy the unmanaged array to managed array.
      Marshal::Copy( (IntPtr) MGRSText, oMGRSString, 0,
                     strlen(MGRSText));

      /// figure out cartesian coord. through geocentric coord.
      double x, y, z;
      Convert_Geodetic_To_Geocentric(LatitudeInRadian,
                        LongitudeInRadian, iAltitude, &x, &y, &z);

      oCartesianX = x;
      oCartesianY = y;
      oCartesianZ = z;
      oAngleWRTNorth = iHeading;
}
```

Figure 5-25 Publish Directed `Conversion Method` for CoordinateConverter

```
/// Conversion method used for subscribe directed Calls
void FOMConverterMgr::CoordinateConverterSubscribeDirectedFunc(
            HLAASCIIstring% oMGRSString, Double% oCartesianZ,
            Double% oCartesianX, Float% oAngleWRTNorth,
            Double% oCartesianY, Float% iLatitude,
            Float% iLongitude, Float% iRoll, Float% iHeading,
            Float% iPitch, Float% iAltitude)
{
      /// Write your conversion code here
      char MGRSString[20];

      /// Convert from MGRS to LLDMS
      double a =  6378137.0;
      double f =  1.0 / 298.257223563;
      char *Ellipsoid_Code = "WGE";
      long status = Set_MGRS_Parameters(a, f, Ellipsoid_Code);

      /// Copy the unmanaged array to managed array.
      Marshal::Copy(oMGRSString, 0, (IntPtr)MGRSString,
            oMGRSString->Length );

      double latitude, longitude;
      status = Convert_MGRS_To_Geodetic(MGRSString, &latitude,
                              &longitude);
      iLatitude  = (float)RADIAN_TO_DEGREE(latitude);
      iLongitude = (float)RADIAN_TO_DEGREE(longitude);

      /// We dont have enough data for roll,
      /// pitch and altitude so a predecided values will be feeded
      iRoll     = 35.0f;
      iPitch    = 45.0f;
      iAltitude = 1000.0f;

      /// We will assign incoming AngleWrtNorth to heading value
      iHeading = oAngleWRTNorth;
}
```

Figure 5-26 Subscribe Directed `Conversion Method` for CoordinateConverter

- War Game Federation

This federation contains federates necessary to simulate a War Game simulation which contains Aircrafts, Air Defense Systems and Environment Federates. In our study, we only make use of Aircraft and Environment federates. Figure 5-5 shows War Game Federation Design model.

- Environment Federate

This federate is responsible from coordinating, as its name implies, environmental activities. These activities are controlling and monitoring of time, date, temperature, fog and wind. The federate is developed using C++\CLI in Visual Studio .NET 2005 with Windows Forms [73] for GUI. The Object Classes with their corresponding `Attributes` is given in Table 5.2. Figure 5-27 shows an example screenshot of this federate. In this screenshot, most functionality needed for a simple federate is illustrated.

The upper left part of form contains information about general federation related data like federate name, FDD file path and RTI parameters. The upper right part of form contains handles (unique identifiers that given by RTI) of classes, interactions, attributes and parameters. The handle values shown with "N\A" means that those attributes or parameters are not used for that federation. There are three important buttons at middle-right part of form called "CREATE/DESTROY/JOIN/LEAVE/INITIALIZE FEDERATION". The "CREATE/DESTROY FEDERATION" creates or destroys federation according to given parameters if it is not created. "JOIN/LEAVE FEDERATION" performs necessary calls to join or leave from federation stated in upper-left part of form with given federate name. "INITIALIZE FEDERATION" button performs necessary HLA initialization according to chosen federate behavior (Publisher or Subscriber). The mentioned elements of user interface are similar for all federates developed, only names are changed, so these are not repeated for other federates. The lower part contains federate specific capabilities. For this federate, Time, Date, Temperature in Celsius, Environment Fog Status and Wind speed with its

direction can be provided to federation for publisher behavior and all these can be observed for subscriber behavior.

Table 5.2 The `Object Class` and `Attributes` for Environment Federate

Class Name	**Attribute Name**
Environment	Temperature
	Time
	Date
	Fog
	WindSpeed
	WindDirection

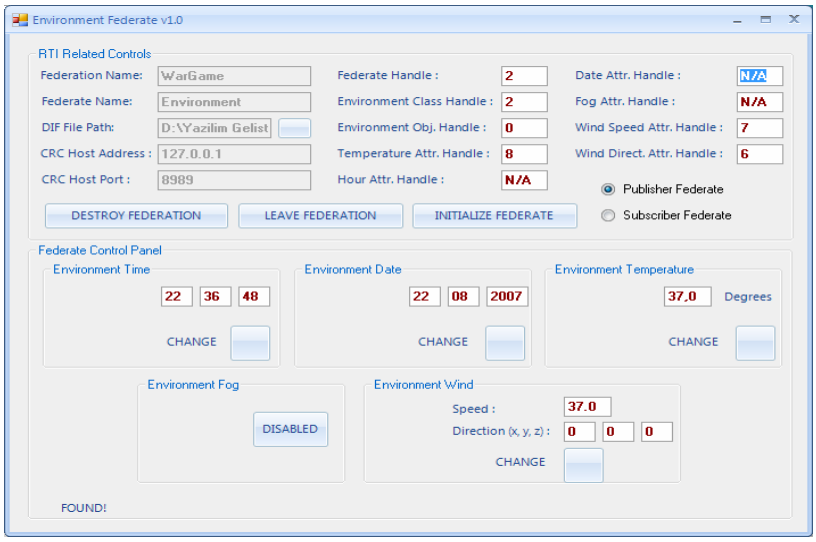

Figure 5-27 Environment Federate GUI

The used Object Classes, Instances, `Attributes` and their handles are shown in GUI.

- Aircraft Federate

This federate is responsible from simulating an aircraft entity. For our study, we only define the coordinate and orientation related features which are used to illustrate many-to-many mapping formation. The `Object Classes` with their corresponding `Attributes` are given in Table 5.3. The federate is developed using C#. Figure 5-28 shows an example screenshot of this federate.

The user interface elements are similar to environment federate, so they are not repeated. The lower part contains federate specific capabilities. For this federate, user can specify the coordinate of aircraft in LLDMS format and also the heading, pitch and roll data can be provided to federation for publisher behavior and all these can be observed for subscriber behavior. Under “CREATE FEDERATION” button, there is also a text box that shows whether a Conversion Component is found or not. This control is reflected to text box after user clicks “CREATE FEDERATION” button.

Table 5.3 The `Object Class` and `Attributes` for Aircraft Federate

Class Name	**Attribute Name**
GPS	Latitude
	Longitude
	Altitude
Compass (HPR)	Heading
	Pitch
	Roll

Figure 5-28 Aircraft Federate GUI

o C3 Federation

This federation contains federates necessary to simulate a rather simplified Command, Control and Communication simulation which contains Command Center, Radar, Targets and Meteorology federates. In our study, we only make use of Target and Meteorology federates. Figure 5-29 shows C3 Federation Design model.

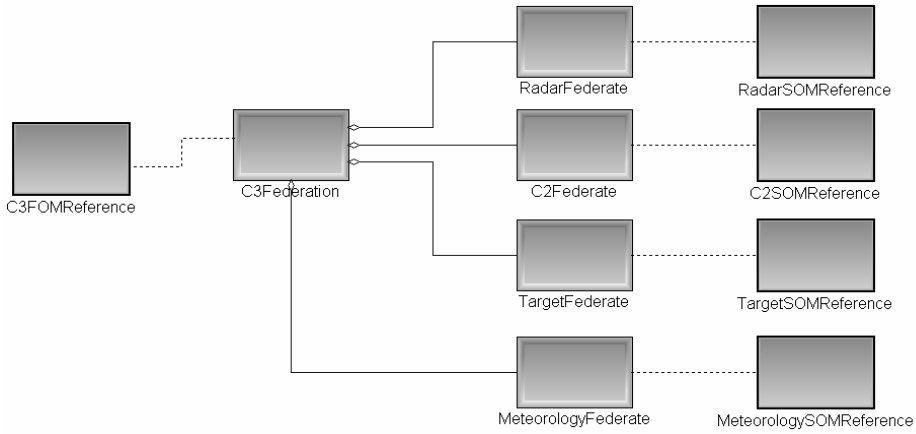

Figure 5-29 C3 Federation Design Model

- Meteorology Federate

This federate is responsible from simulating meteorology related activities that are involved for a C3 simulation like. The `Object Classes` with their corresponding `Attributes` are given in Table 5.4. The federate is developed using C#. Figure 5-30 shows an example screenshot of this federate.

The upper part of federate contains similar information to previous federates, so they are not repeated here. The lower part contains federate specific capabilities. For this federate, user can specify humidity, atmosphere pressure, temperature and wind related parameters here. These data can be provided to federation for publisher behavior and all these can also be observed for subscriber behavior.

Table 5.4 The `Object Class` and `Attributes` for Meteorology Federate

Class Name	Attribute Name
Meteorology	AverageTemperature
	Humidity
	SpeedOfWind
	DirectionOfWind
	WindSpeedCategory
	AtmospherePressure

Figure 5-30 Meteorology Federate GUI

- Target Federate

This federate is responsible from simulating an entity which can be a ground, sea or air unit that is detected by the radars in C3 simulation. The `Object Classes` with their corresponding `Attributes` are given in Table 5.5. The federate is developed using C++\CLI. Figure 5-31 shows an example screenshot of this federate.

The upper part of federate contains similar information to previous federates, so they are not repeated here. The lower part contains federate specific capabilities. For this federate, user can specify Cartesian coordinates and angle with respect to north under "Target Detailed Data" control panel and coordinates of target in MGRS position. The interaction target hit status can also be stated here. All these data can be provided to federation for publisher behavior and can also be observed for subscriber behavior.

Table 5.5 The `Object Class` and `Attributes` for Target Federate

Class Name	**Attribute Name**
DetailedTargetData	CartesianX
	CartesianY
	CartesianZ
	AngleWithRespectToNorth
Coordinate	MGRSCoordinateString

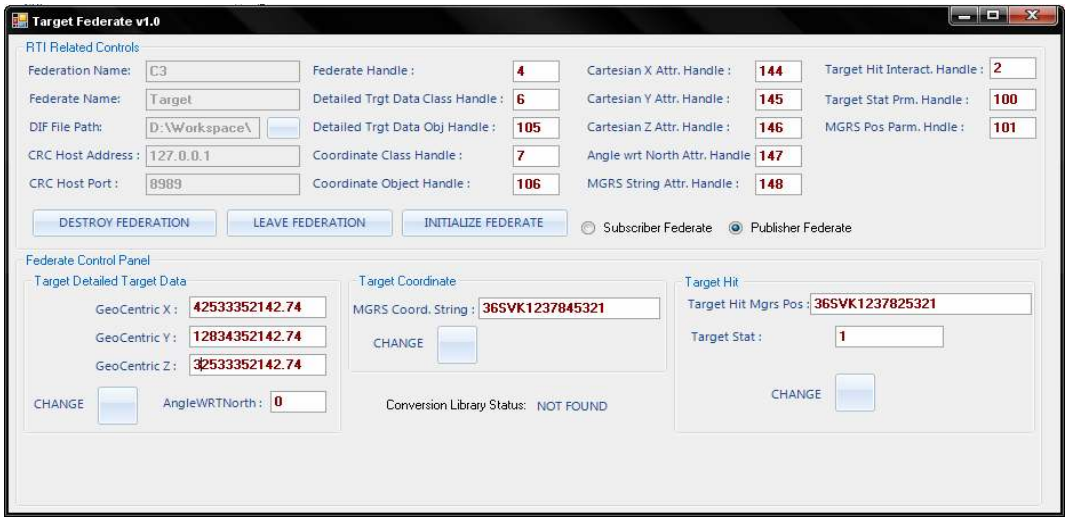

Figure 5-31 Target Federate GUI

The mentioned example illustrated only one possible usage of this solution for FIP. There are also some other possible scenarios that make use of libraries mentioned in previous sections as illustrated in Figure 5-32.

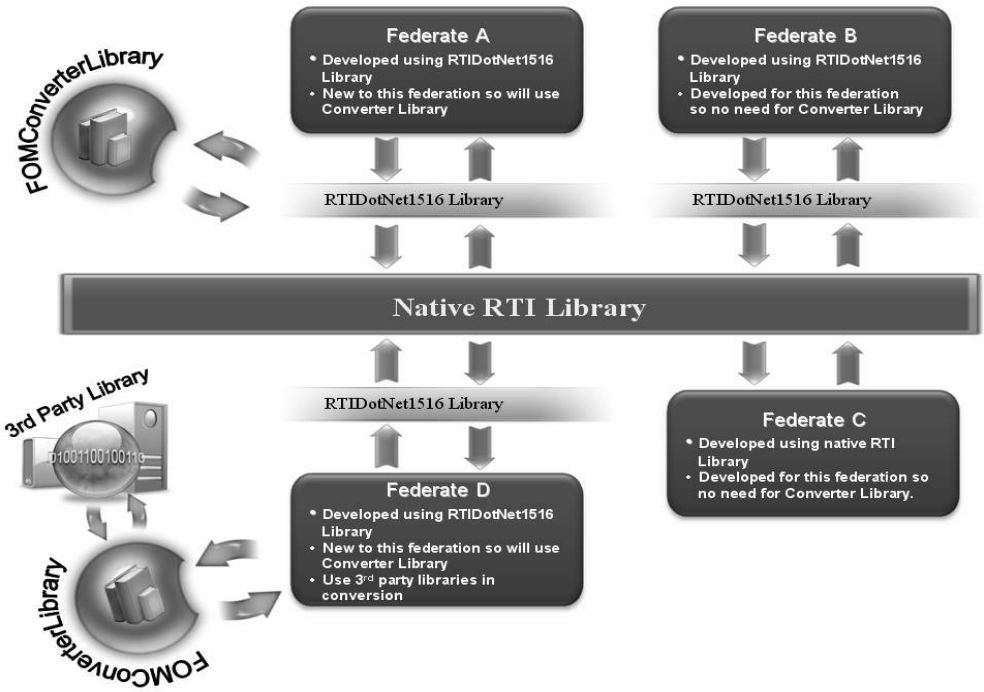

Figure 5-32 An example HLA Federation that shows possible scenarios

Four federates are used to illustrate the possible scenarios. Although it is not shown, federates using `RTIDotNet1516` library can be in any .NET compatible programming languages. The federate A shows a typical usage of `RTIDotNet1516` library and `Conversion Component (FOMConverter Library)`. It is not developed especially for this federation, so it uses `FOMConverter` library for conversions. The federate B developed for this federation using `RTIDotNet1516` library and do not need to use any conversion related routines, consequently not using any `Conversion Component`. The federate C developed again for this federation in native RTI library so not using both of `RTIDotNet1516` and `Conversion Component`. The federate D shows a possible usage of 3rd party libraries with `RTIDotNet1516` and `FOMConverter` Library. These 3rd party libraries can be .NET compatible or non .NET compatible libraries.

CHAPTER 6

DISCUSSION AND CONCLUSION

6.1 Achievements

In this study, a Model Based Solution which employs MIC and .NET 2.0 technologies is analyzed to solve a well-known FIP that frequently encountered in HLA compliant distributed simulation applications.

First of all, previous approaches are studied by examining what they provide or lack and than their comparison have been done with proposed solution. After that, the technologies and tools are examined with their role in overall solution. The problem is described and solution is defined by grouping related activities into three well-defined phases which are modeling, automatic code and component generation. While describing problem and solution, new supporting definitions like dependency levels and mapping formations are given and how proposed solution handle these cases are illustrated by examples. Finally, case study that illustrates the practical usage of the solution and approach is described.

The modeling environment depends on FIM which is derived from previously developed HOMM by adding new elements to solve FIP. GME is used as meta-programmable modeling tool for all Metamodeling and Modeling activities.

In this approach, not only the FIP solution is provided, but also benefits of MDSD and user friendly Modeling Environment of GME is offered to developer with .NET 2.0 technologies. The `RTIDotNet1516` library helps developer to develop IEEE 1516 compatible platform and programming language independent .NET

federate applications. In this study, automatically generated code is fully complying with rules defined in CLS, so that it can be used from any other .NET compatible programming languages.

6.2 Limitations of Current Work

There are also some limitations which are not handled in this study. The first one is the Time, Data Distribution and Ownership Management services. These services are usually executed during federation execution and modeling of these services and behaviors are difficult to model so they are not handled in this study. As a result of this, developer currently cannot use related services.

The second one is the assumption of proper development of federate code. In other words, developer is expected to develop its federate code so that necessary attributes/parameters are provided to other federations. For instance, if user want to use Attribute A and B to as Attribute C for Destination Federation through a `Conversion Method`, but do not publish/update one of Attribute A or B, the Attribute C will not be represented properly in Target Federation.

The other one is about programmatic details like multidimensional arrays. The current metamodel does not provide necessary information about sizes of multidimensional arrays.

In addition to all these, there are also some other limitations which prevent us to solve the FIP with a hundred percent and can not be tackled for the time being. The most important one is the fully automatic solution. The current approaches and our solution need developer to define, at least, the relationship among Source and Target Federation by hand. In other words, a mapping or Correspondence Model for our case cannot be generated automatically from given new FOM. This is mainly stemmed from the fact that each federation has its own semantics and for a possible mapping a manual interference is needed. The additional function or service calls are also unavoidable. Although this can be minimized by embedding this into core RTI, there will be still some overhead. This is done with

`RTIDotNet1516` library in this study which can be seen as an RTI Abstraction layer.

6.3 Future Work

Based on findings from this study, this approach can successfully be adapted for many forthcoming Modeling Driven HLA Application development tools. To make use of this approach tools must adopt our Metamodel, FIM. The approach might also be improved by employing some other behavioral modeling approaches for model conversion routines. Moreover this approach could easily be applied to other distributed simulation standards or software problems.

Some other obvious improvements are generation of codes in other .NET compatible programming languages and porting solution to Java or native C++ platforms and providing a higher level API that hides HLA details from developer who are not familiar to HLA.

REFERENCES

[1] IEEE Std. 1516, "*IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules",* IEEE, 2000.

[2] IEEE Std. 1516.1, "*IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification",* IEEE, 2000.

[3] IEEE Std. 1516.2, "*IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template Specification",* IEEE, 2000.

[4] IEEE Std. 1516.3, "*IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP)",* IEEE, 2003.

[5] IEEE web site*, "The world's leading professional association for the advancement of technology",* http://www.ieee.org, last accessed 18.07.2007.

[6] Test and Training Enabling Architecture (TENA), *"Home – TENA: Introduction – TENA SDA Website",* https://www.tena-sda.org/display/intro/Home*,* last accessed 18.07.2007.

[7] DMSO web site*, "Defense Modeling and Simulation Office",* https://www.dmso.mil, last accessed 18.07.2007.

[8] IEEE, *"IEEE Begins to Revise Four Simulation Standards",* http://standards.ieee.org/announcements/pr_simulation.html, last accessed 10.06.2007.

[9] Granowetter, L., "*Solving the FOM-Independence Problem",* 1999 Spring Simulation Interoperability Workshop (SIW).

[10] Cetinkaya D., Oguztuzun H., "*A Metamodel for the HLA Object Model",* 20th European Conference on Modeling and Simulation (ECMS), pp 207-213, 2006.

[11] Ledeczi A., Maroti M., Bakay A., Karsai G., Garrett J., Thomason C., Nordstrom G., Sprinkle J., Volgyesi P., *"The Generic Modeling Environment",* IEEE International Workshop on Intelligent Signal Processing (WISP'2001)*,* Budapest, Hungary, 2001.

[12] Microsoft web site, *"Visual Studio 2005 Developer Center",* http://msdn2.microsoft.com/en-us/vstudio/default.aspx, last accessed 18.07.2007.

[13] MÄK Technologies, Inc., "*MÄK Technologies, Inc.",* https://www.mak.com, last accessed 05.06.2007.

[14] MÄK Technologies, "*VR-Link®"*, http://www.mak.com/products/vrlink.php, last accessed 09.06.2007.

[15] Graham J., Foscue J., Cutts D., *"HLA Object Models as Software Object Models: An approach to Automatic Software Generation from HLA Object Models",* 1998 Spring Simulation Interoperability Workshop (SIW), 1998.

[16] Calytrix Technologies, "*SIMplicity",* http://simplicity.calytrix.com, last accessed 01.08.2007.

[17] Çelik T., Sütbaş R., İmre K., *"HLA için Modelleme, Otomatik Kod Üretme, İzleme ve Sınama Araçları",* USMOS 2005, Ankara, Turkey, 2005.

[18] AEgis Technologies, "*AEgis Technologies Modeling and Simulation Advanced small business for modeling and simulation Huntsville AL",* http://www.aegistg.com, last accessed 26.07.2007.

[19] Hunt K., Graham J., "*Using OMni as an Interface to the HLA RTI*", http://www.hlalabworks.com/papers/OMni-WP.pdf, last accessed 26.08.2006.

[20] Simulation Interoperability Standards Organization, "*SISO Standards",* http://www.sisostds.org/index.php?tg=fileman&idx=get&id=5&gr=Y&path=SISO+Products%2FSISO+Standards&file=GRIM_RPR-FOM_1-0v2.pdf , last accessed 18.07.2007.

[21] IEEE Std. IEEE 1278.1, "*IEEE Standard for Distributed Interactive Simulation - Application Protocols",* IEEE, 1995.

[22] IEEE Std. IEEE 1278.1a, "*Supplement to IEEE Std 1278.1-1995*, *IEEE Standard for Distributed Interactive Simulation - Application Protocols",* IEEE, 1998.

[23] Möller B., Löfstrand B., Karlsson M., *"An Overview of the HLA Evolved Modular FOMs",* 2007 Spring Simulation Interoperability Workshop (SIW), 2007.

[24] Kuhl F., Weatherly R., Dahmann J., "*Creating Computer Simulation Systems: An Introduction to the High Level Architecture",* Prentice Hall, New Jersey, 1999.

[25] Shaw M., Garlan D., *"Software Architecture: Perspectives on an Emerging Discipline"*, Prentice-Hall, New Jersey, 1996.

[26] Fujimoto R. M., "*Parallel and Distributed Simulation Systems"*, Wiley, New York, 2000.

[27] OMG web site*,"Object Management Group"*, http://www.omg.org, last accessed 06.06.2007.

[28] Simulation Interoperability Standards Organization, "*SISO"*, http://www.sisostds.org/index.php, last accessed 18.07.2007.

[29] B´ezivin J., Gerbé O., *"Towards a Precise Definition of the OMG/MDA Framework"*, ASE'01, Automated Software Engineering, San Diego, USA, November 26-29, 2001.

[30] Völter M., Stahl T., Bettin J., Haase A., Helsen S., Czarnecki K., "*Model-Driven Software Development"*, Wiley, 2006.

[31] Özhan, G., Oğuztüzün, H., "*Model-Integrated Development of HLA-Based Field Artillery Simulation"*, 2006 European Simulation Interoperability Workshop (SIW), Sweden, 2006.

[32] Tolk A., "*Avoiding Another Green Elephant –A Proposal for the Next generation HLA based on the Model Driven Architecture"*, 2002 Fall Simulation Interoperability Workshop (SIW), 2002.

[33] Tolk A., "*Metamodels and Mappings – Ending the Interoperability War"*, 2004 Fall Simulation Interoperability Workshop (SIW), 2004.

[34] Parr S., Keith-Magee R., "*Making the Case for MDA"*, 2003 Fall Simulation Interoperability Workshop (SIW), 2003.

[35] Kleppe A., Warmer S., Bast W., "*MDA Explained - The Model Driven Architecture: Practice and Promise"*, Addison-Wesley, Boston, 2003.

[36] OMG web site*, "MDA Guide v1.0.1"*, http://www.omg.org/docs/omg/03-06-01.pdf, last accessed 06.06.2007.

[37] OMG web site, "*MDA FAQ"*, http://www.omg.org/mda/faq_mda.htm, last accessed 10.06.2007.

[38] Czarnecki K., Helsen S., *"Classification of Model Transformation Approaches"*, OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, Anaheim, California, USA, 2003.

[39] Mens T., Czarnecki K., Gorp P. V., *"A Taxonomy of Model Transformations"*, Language Engineering for Model-Driven Software Development, Dagstuhl, Germany, 2004.

[40] Czarnecki K., Helsen S., *"Feature-based survey of model transformation approaches",* IBM Systems Journal, vol. 45, no. 3, pp 621-645, 2006.

[41] Voelter M., *"A Catalog of Patterns for Program Generation"*, Eighth European Conference on Pattern Languages of Programs, Irsee, Germany, 2003.

[42] OMG Web Site, *"Object Management Group: MOF QVT Final Adopted Specification",* http://www.omg.org/docs/ptc/05-11-01.pdf, last accessed 18.07.2007.

[43] Greenfield J. and Short K., *"Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools",* John Wiley and Sons, Indianapolis, 2004.

[44] Nordstrom G., Sztipanovits J., Karsai G., Ledeczi A., "*Metamodeling – Rapid Design and Evolution of Domain-Specific Modeling Environments",* IEEE ECBS'99 Conference, Tennessee, 1999.

[45] Karsai G., Agrawal A., *"Graph Transformations in OMG's Model-Driven Architecture",* Applications of Graph Transformations with Industrial Relevance International Workshop, Virginia, 2003.

[46] Karsai G., Agrawal A., Ledeczi A., *"A Metamodel-Driven MDA Process and its Tools," in WISME, UML 2003 Conference,* San Francisco, 2003.

[47] Sudarsan R., Gray J., *"Meta-Model Search: Using XPath to Search Domain-Specific Models",* International Conference on Software Engineering Research and Practice, Nevada, 2005.

[48] Gray J., Bapty T., Neema S., Tuck J., *"Handling Crosscutting Constraints in Domain-Specific Modeling",* Communications of the ACM Journal, vol. 44 no. 10, pp. 87-93, 2001.

[49] Consel C., Marlet R., *"Architecturing software using a methodology for language development",* 10th International Symposium on Programming Language Implementation and Logic Programming, Pisa, Italy, September, 1998.

[50] Karsai G., Agrawal A., Shi F., Sprinkle J., *"On the Use of Graph Transformations for the Formal Specification of Model Interpreters",* Universal Computer Science Journal, vol. 9, no. 11, pp. 1296-1321, 2003.

[51] OMG web site, "*UML Infrastructure v2.1.1"*, http://www.omg.org/cgi-bin/apps/doc?formal/07-02-06.pdf, last accessed 06.06.2007.

[52] Karsai G., Maroti M., Ledeczi A., Gray J., Sztipanovits J., "*Composition and Cloning in Modeling and Meta-Modeling",* IEEE Transactions on Control System Technology, vol.12 no.2, pp. 263-278, 2004.

[53] OMG Web Site, *"OMG Document"*, http://www.omg.org/cgi-bin/doc?formal/01-09-67, last accessed 18.07.2007.

[54] OMG Web Site, *"Meta-Object Facility (MOF™), version 1.4"*, http://www.omg.org/technology/documents/formal/mof.htm, last accessed 18.07.2007.

[55] Institute for Software Integrated Systems Vanderbilt University, "*GME 6 User's Manual Version 6.0",* pp 12, Vanderbilt University, 2006.

[56] Emerson M.J., "*GME-MOF: An MDA Metamodeling Environment for GME",* MSc Thesis in Computer Science at Vanderbilt University, Tennessee, 2005.

[57] Magnetar Games, *Chronos,* http://www.magnetargames.com/Products/Chronos, last accessed 10.06.2007.

[58] Sutter H., *"A Design Rationale for C++\CLI"*, http://www.gotw.ca/publications/C++CLIRationale.pdf, 2006.

[59] ISO, *"ISO/IEC 2327, Common Language Infrastructure*" http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=42927, last accessed 10.06.2007

[60] Mono, *"Mono Project"*, http://www.mono-project.com, last accessed 09.05.2007.

[61] DotGNU Portable.NET, *"DotGNU Project"*, http://dotgnu.org/pnet.html, last accessed 09.05.2007.

[62] Microsoft, *".NET Compact Framework"*, http://msdn2.microsoft.com/en-us/netframework/aa497273.aspx, last accessed 10.06.2007.

[63] StartVbDotNet.com, *".NET Framework Supported Languages",* http://www.startvbdotnet.com/dotnet/languages.aspx, last accessed 18.07.2007.

[64] Perry S. C., *"Core C# and .NET"*, Prentice Hall PTR, 2005.

[65] ECMA International, "*Standard ECMA-335: Common Language Infrastructure (CLI)"*, http://www.ecma-international.org/publications/standards/Ecma-335.htm, last accessed 10.06.2007.

[66] Abrams B., *".NET Framework Standard Library Annotated Reference, Volume 1/2 Base Class Library and Extended Numerics Library",* Addison Wesley, 2004.

[67] Meijer E., Gouh J., "*Technical Overview of the Common Language Runtime*", http://research.microsoft.com/~emeijer/Papers/CLR.pdf, last accessed 10.06.2007.

[68] Fraser S. R. G., *"Pro Visual C++ CLI and the .NET 2.0 Platform"*, Apress, New York, USA, 2005.

[69] Microsoft Web Site, "*NGEN Tool"*, http://msdn.microsoft.com/msdnmag/issues/05/04/NGen/default.aspx, last accessed 09.05.2007.

[70] ECMA International, "*Standard ECMA-372: C++/CLI Language Specification"*, http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-372.pdf , last accessed 18.07.2007.

[71] Lippmann S., *"Pure C++: Hello, C++/CLI",* MSDN Magazine, February 2005.

[72] Szyperski C., "*Component Software: Beyond Object-Oriented Programming*", ACM Press and Addison-Wesley, New York, 1998.

[73] Microsoft Web Site, *"Windows Forms",* http://msdn2.microsoft.com/en-us/netframework/aa497342.aspx, last accessed 18.07.2007.

[74] Microsoft Web Site, *"MFC Reference (MFC)",* http://msdn2.microsoft.com/en-us/library/d06h2x6e(VS.80).aspx, last accessed 18.07.2007.

[75] Pitch Technologies*, "Pitch RTI (pRTI)",* http://www.pitch.se, last accessed 01.08.2005.

[76] SGI, "*Standard Template Library Programmer's Guide",* http://www.sgi.com/tech/stl/, last accessed 18.07.2007.

APPENDIX A

USER'S GUIDE

This is the User's Guide of the `Correspondence Model` Design Environment (CMDE) which is automatically generated by GME according to FIM. It explains how to use the GME to construct Corresponding Models among Source and Target Federation object models.

To be able to use the CMDE, first of all FIM should be registered. Open GME, click on File Menu and then click Add from File button on Select Paradigm dialog and then select FIM metamodel .xmp file which will register the metamodel.

In this User's Guide, creating a new HLA OMT project, Federation Model and Object model is not described which can be found in [10] with details. In this guide, the steps necessary to construct a `Correspondence Model` and execution of `Interpreter` is described.

To construct `Correspondence Models`, two federation models are needed one which will belong to Source Federation and other will belong to Target Federation. These two models can be constructed from scratch using CMDE by separating modeling elements with Folders.

Two federation models can also be imported into one Modeling Environment which is usually the more preferable for already modeled federations. To do so, a new CMDE project can be created and then by using File/Import XML utility the other two Federation Models can be imported into the same Modeling Environment or an existing model can be opened then the model can be imported

into same Modeling Environment by following just mentioned procedure. Here there is one point need to be taken into consideration, when these two models are imported into same modeling environment, there will be two IEEE 1516 library which is same with each other, so one of this libraries can be deleted by hand. There could also be inconsistencies with data types of these two federation model which can also be merged to one Data Type folder to prevent these.

After Source and Target Federation Models are present in the same modeling environment, the `Correspondence Models` can be constructed. First, the related Object Model Folder, then FOM Model and finally the Objects Model (Interaction folder for Interaction `Correspondence Models`) are opened. Under this model, the Objects or `Interactions` Object Oriented hierarchy can be seen. The steps necessary to go related model where `Correspondence Model` is constructed shown in Figure A-1 below;

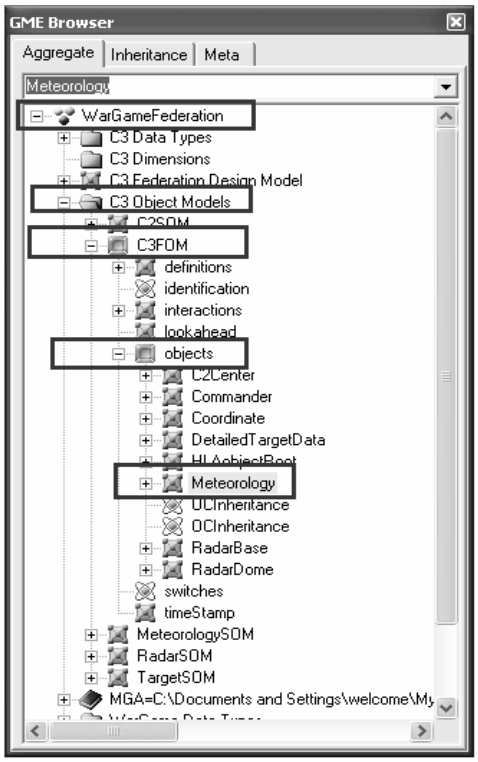

Figure A-1 Steps necessary to go related model where `Correspondence Model` constructed

The object that will be used for Correspondence Model is opened than. The Meteorology Object Class is selected in Figure A-1 shown above. When this model is opened, the Attributes owned by this class are shown. To construct a Correspondence Model, at least two Attributes one which comes from Source Federate and the other from Target Federate and one Converter is needed. Converter atoms can be dragged into Modeling Environment from Part Browser which is shown in Figure A-2 for Attributes.

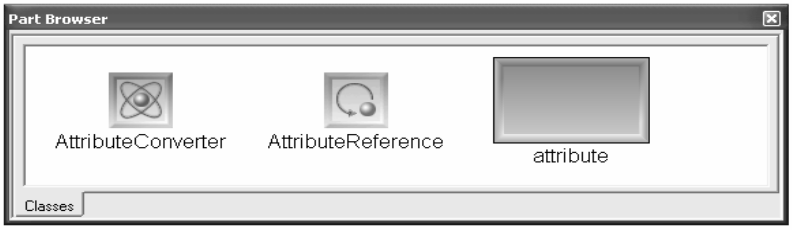

Figure A-2 The Part Browser Window for Attributes

One final element necessary to construct Correspondence Model is other Attribute that will be used with Converter. To bring other Attributes, the provided AttributeReference element can be used. For each Attribute that will be brought into environment one of these AttributeReferences should be dragged. After dragging these elements, the only thing that needs to be done is drag corresponding Attribute from Browser tree onto AttributeReference.

When Attributes or Parameters and Converters are brought into environment, the final step is to connect these items with each other and make necessary settings to Converters. The connections are built using the interface provided by GME. The restrictions related with these connections are described in section 5.2.1.1 . To change the attributes of Converters, the Object Inspector Dialog (View->Attribute Panel) can be used which is shown in Figure A-3.

After all `Correspondence Models` are constructed, automatic code generation can be initiated. This can be done by clicking `Interpreter` icon on toolbox of GME which is shown in Figure C-1. There are also some settings need to be done on `Interpreter` dialog which is grouped into Control Panel bottom right side of dialog as shown in Figure A-4.

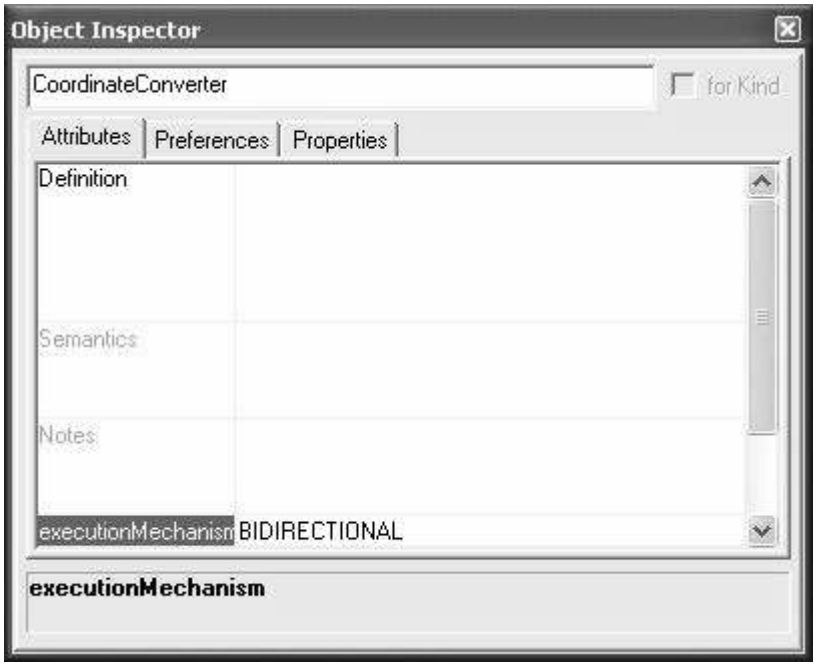

Figure A-3 The Object Inspector Window for Converter attributes

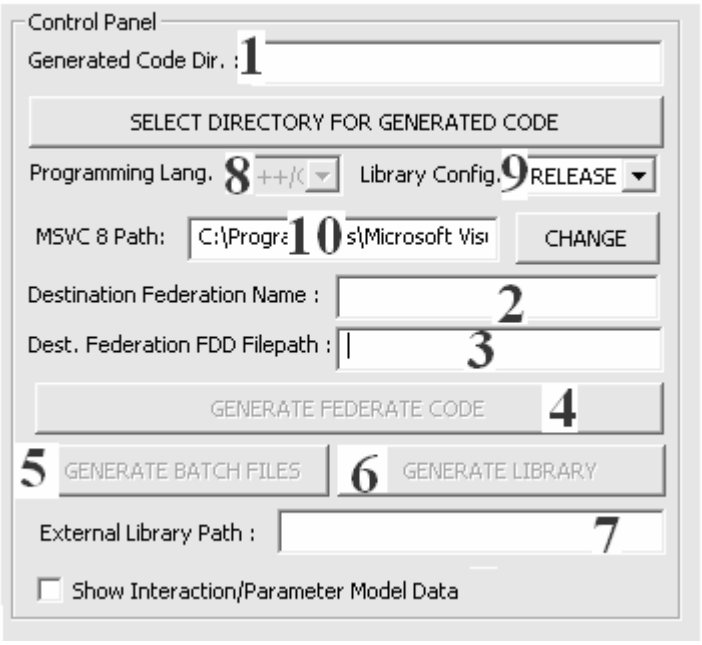

Figure A-4 The Control Panel of `Interpreter`

The control marked by 1 show the directory that conversion library code going to be generated. The Target Federation name and FDD file that will be used by converter library for Source Federation are entered through marking 2 and 3. Marking 4 is enabled after developer enter directory for code that going to be generated. These four controls are enough for automatic code generation. The rest of the controls are put for developers who do not have Integrated Development Environments and use makefiles. The button shown by marking 5 generates makefiles to directory shown by control 7 according to entered compiler path in marking 10 and compiler preference in marking 9. The marking 8 is put for selection of programming language which currently C++\CLI by default.

APPENDIX B

THE IMPLEMENTED RTI SERVICES FOR FIM

Table B.1 The implemented RTI services

The Service Group Name	**The Service Name**
Federation Management Services	All services are implemented according to FIP.
Declaration Management Services	All services are implemented according to FIP.
Object Management Services	One of each UpdateAttributeValues, SendInteraction and deleteObjectInstance overloaded methods are implemented. The other methods are containing arguments related with Time and Data Distribution Management so not implemented.
Ownership Management Services	Not interfered.
Time Management Services	Not interfered.
Data Distribution Services	Not interfered.
Support Services	getObjectClassHandle, getObjectClassName, getAttributeHandle, getAttributeName, getInteractionClassHandle, getInteractionClassName, getParameterHandle, getParameterName, getObjectInstanceHandle, getObjectInstanceName, getKnownObjectClassHandle, getTransportationType, getTransportationName, getOrderType, getOrderName services are implemented according to FIP.

APPENDIX C

INTERPRETER SOFTWARE

In this chapter, the technical information about the `Interpreter` which is used to automatically generate code is given. As stated in MIC, the interpreters are used to perform semantic translations by using models to generate executable models or a program which is source code in our case. The most important activities performed by `Interpreter` are briefly analysis of model constructed, programmatically verification of models and finally automatic generation of source code. The `Interpreter` is add-in software that can be integrated into GME modeling environment through plug-in mechanism provided by again GME. Figure C-1 illustrates how interpreter is seen in modeling environment.

Figure C-1 Illustration of `Interpreter` software in modeling environment

To develop an `Interpreter` first a tool that comes with GME, Component Configurator, is used to create necessary files needed for `Interpreter` development. These files are configuration, Microsoft Foundation Classes (MFC) [74] project and `Interpreter` Common files. The `Interpreter` is developed by using these and additional files supplied by developer. It does not need to have a Graphical User Interface (GUI), but GUI can also be added. Some GUI elements are added into our `Interpreter` to make it easy for use.

The Figure C-2 shows our Interpreter's GUI that developer see when she press the button shown in Figure C-1. The interpreter is explained through the red markings made on Figure C-2. The list in marking 1 show the Attribute and Parameter converters defined in Model. According to selection from this list, input and output Attribute or Parameter and corresponding arguments names are listed in lists shown in marking 2 and 3. The corresponding owner Object Class or Interaction of these input and output Attributes or Parameters are shown in lists illustrated by marking 4 and 5. The panel shown in marking 6 gives detailed information collected about selected input or output Attribute or Parameter lists like their name, type, owner Object Class or Interaction and data type. The list shown by marking 7 lists all data types of Attribute or Parameters related in Correspondence Model with their alias and data type. The panel which is illustrated with marking 8 shows warnings, errors and state of Interpreter execution. The panel illustrated with marking 9 shows the information collected about SynchronizationPointMapper. The panel illustrated by marking 10 contains controls about automatic code generation like where the code will generated, Target Federation name and FDD file path text boxes that will be used during Federation Execution. In addition to these, other controls for generation of makefiles and compilation of Converter Library are provided in this panel.

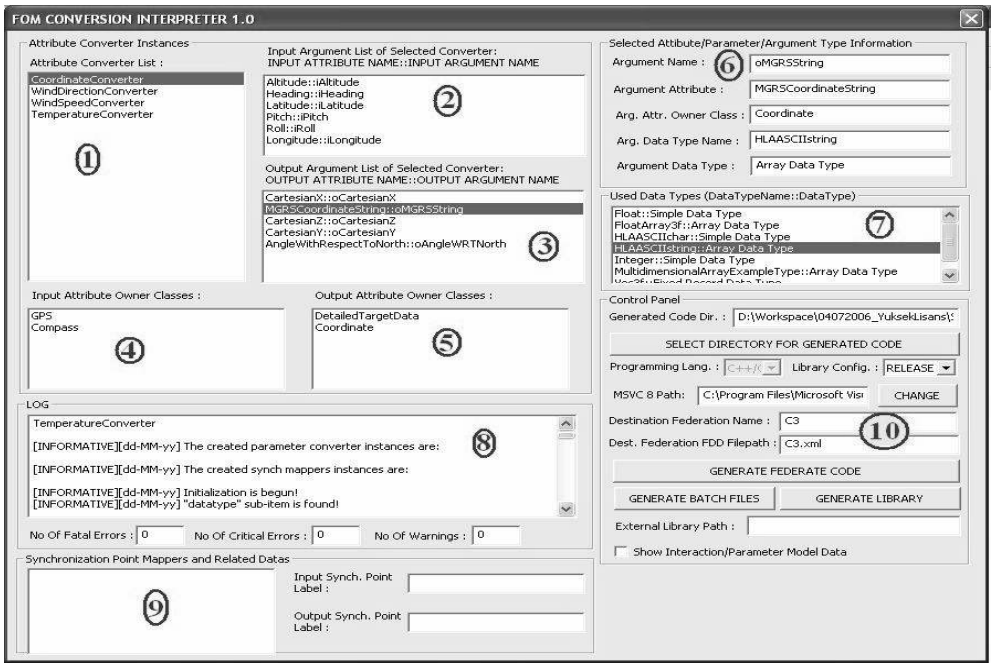

Figure C-2 A screenshot of FOM Conversion `Interpreter`

This software developed using MS Visual Studio .NET 2003 and MFC in C++.

APPENDIX D

RTIDOTNET1516 LIBRARY

There are two important libraries developed in the scope of this study; one of them, the `FOMConverter` Library as `Conversion Component` is described in section 5.2. In this section, the other important library which is `RTIDotNet1516` is described. As `FOMConverter` library this is also developed using .NET 2.0 assembly technology. This library can be seen as an extensive wrapper to underlying native RTI library which not only provide us a .NET 2.0 compatible RTI interface but also an abstraction layer for FIP. After all, this can also be considered as an API which is mainly used by federate developers to develop HLA and .NET 2.0 compliant applications.

The most important reason to develop this library is the lack of .NET 2.0 compatible RTI software. In fact, it appears to be there exists only a few RTI implementations for .NET, in fact only one of them is found and evaluated [16], and nevertheless it is based on .NET 1.1. So this library is developed from scratch in .NET 2.0.

This library is developed using new C++\CLI programming language which not only provide us to use the power of native C++ but also help us to easily port the native RTI code to .NET world. The detail of this programming language is given in section 4.2.

The developed wrapper is uses Pitch Technologies' Portable RTI (pRTI) Light Edition as underlying RTI library [75]. The other option was MÄK Technologies' MÄK RTI. The MÄK is not chosen, because there is a restriction on the number of federates which is only two and also no GUI that can help us to monitor

federation is provided. We choose pRTI implementation, because the evaluation version provides us a GUI in which the activities in Federation Execution can easily be observed which is very critical for us. Moreover no limit on the number of federates for a federation is exists. However, it also has some restrictions that trouble us during development which is usually based on the number of services called during federation execution and when this number is reached an RTI exception is thrown and execution of federation is terminated.

The most important capabilities of this assembly are being an abstraction layer for native RTI library by providing same service interfaces to developer and containing base classes for `FOMConverter` Library mentioned in component generation phase. While developing this library, special attention is paid to preserve same interface with RTI services to prevent developers of this library to learn new API which is the case for MÄK's VR-Link tool as mentioned in chapter 1. We also do not develop services unrelated with our study which is mentioned in section 2.3.

As developing `RTIDotNet1516` library, the original method prototypes in native pRTI library kept same. The data types, enumerations, exceptions and class hierarchy used in native pRTI library is also ported to .NET 2.0. The `RTIDotNet1516` contains two groups of classes. The first group of classes is mainly responsible from implementation pRTI services and necessary data types. The second group of classes is responsible from providing base classes and data types to `FOMConverter` library.

As stated above one group of classes and related constructs that developed are responsible from implementation of pRTI. In this group, for each native pRTI classes, a managed class that prefixed with "RTIDotNet" is developed. For instance, for RTIambassador native class, RTIDotNetRTIambassador class is developed. The other wrapped classes that similarly developed are FederateAmbassador, LogicalTime, OrderType, RangeBounds, ResignAction, RestoreFailureReason, RestoreStatus, RTIambassadorFactory, SaveFailureReason SaveStatus, ServiceGroupIndicator, SynchronizationFailureReason, Trans-

portationType, VariableLengthValueClass and all other Value, Handle, HandleSet and HandleValueMap classes defined using VariableLengthValueClass. A declaration of developed RTIDotNetRTIambassador class is given in Figure D-1.

```
public ref class RTIDotNetRTIambassador
{
	/// This member used to hold the conversion library loaded
	/// dynamically
	Assembly^ mConverterAssembly;

	/// Conversion library handle
	FOMConverterMgrBase^ mConverterLibrary;
	...

	/// Reference to Native RTI ambassador
	RTI::RTIambassador* mNativeMember;
	AutoWrapperClass<RTI::RTIambassador>* mRTIWrapper;
	...
public:
	/// Constructor
	RTIDotNetRTIambassador(
		std::auto_ptr< RTI::RTIambassador > iArg,
		bool iIsConversionLibraryGiven,
		String^ iConversionLibraryName,
		String^ iConversionLibraryPath);
	...
	/// Destructor
	~RTIDotNetRTIambassador();

	/// Finalizer
	!RTIDotNetRTIambassador();

	/// 4.2 Federation Management service
	virtual void createFederationExecution
		(String^ federationExecutionName,
		String^ fullPathNameToTheFDDfile);
	...
};
```

Figure D-1 Snapshot for Configuration Arguments

Exceptions which defined as different classes in native RTI library are implemented as one .NET managed class with enumerated exception types to make usage of exception mechanism .NET compatible, easy and more user-friendly. A code portion related with exceptions is given in Figure D-2.

```
/// Enumaration that define RTI exception types
public enum class ExceptionType
{
      AsynchronousDeliveryAlreadyDisabled,
      AsynchronousDeliveryAlreadyEnabled,
      AttributeAcquisitionWasNotCanceled,
      ...
};

/// Managed exception class
public ref class RTIDotNetException : Exception
{
public:
      /// Exception type
      ExceptionType mType;

      /// Reference to original exception
      RTI::exception* mException;

      /// Property to get type of exception
      property ExceptionType TypeOfException
      {
            ExceptionType get() { return mType; }
      }
      ...
 };
```

Figure D-2 A code portion related with exceptions

The native pRTI make extensive use of the STL (Standard Template Library) [76] container types, we port them to .NET 2.0 by making use of corresponding .NET generic types (which are in fact introduced with .NET 2.0). For example, List<> generic type used for std::set, std::vector and Dictionary<> generic type used for std::map.

Finally, original namespace “RTI” changed to “RTIDotNetLibrary1516” to prevent any confusion. However, original RTI namespace is still accessible which let C++\CLI like native accessible .NET programming languages make use of this interface and corresponding constructs.

The second group of classes is related with conversion operations. Most of these classes are used for representing the model that developer created as described in first title. Some of these are “HLAAttribute”, “HLAClass”, “HLAClassObject”,

“HLAInteraction” and “HLAParameter” which represents the state of corresponding HLA constructs. The `AttributeConverter` and `ParameterConverter` classes are basic classes that perform conversions at lowest level during federation execution. There is also a `FOMConverter-MgrBase` abstract class which is responsible from controlling overall execution of conversion calls and callbacks through inherited `FOMConverterMgr` class.

As a result of using C++\CLI for developing `RTIDotNet1516` library make it more rigid and effective in terms of performance rather than using marshalling calls or p/invoke calls from C#/VB.NET which was in fact the only way of porting a native library to .NET world.

APPENDIX E

AUTOMATICALLY GENERATED FILES

As discussed in 5.2.2.2 , the Interpreter software generates more than six files in automatic code generation phase. These files are described below in detail with snapshots.

- **"ConverterLib.h"**

This header file contains data type declarations about `Attributes` and `Parameters` used in Correspondence map at modeling phase. These data types is generated and provided to developer to be used in `Conversion Methods` to prevent her from dealing with raw byte arrays and type conversion. The `Interpreter` can produce Simple Data Types (e.g. Int16, Int32, Single, Double), Enumerated Data Types (e.g. Boolean), Array Data Types (e.g. HLAASCIIstring) and Fixed Record Data Types (e.g. GPS, Vec3f) with two groups of supporting methods, which can be used in `Conversion Methods` or can be used in Federate Code directly. These supporting methods are given below.

GetBytes, SetBytes, GetBytes*ArrayTypeName,* SetBytes*ArrayTypeName:*
These are used to create a new instance of corresponding type from its byte-wise representation and get byte representation of related data type. They also used in `FOMConverterMgr` library internally in byte conversions for RTI service calls. These methods are generated for Fixed Record Data and Array Data Types.

GetSize, GetSize*ArrayTypeName:*
These methods are used to calculate the size of a Fixed Record Data or Array Data types in bytes.

In this file, a header of `FOMConverterMgr` class that responsible from holding information necessary for conversion, performing conversions and related initializations is also generated. For each of `AttributeConverter` and `ParameterConverter`, member methods like, "*AttributeName*Publish DirectedFunc", "*InteractionName*SubscribeDirectedFunc" prototypes are generated according to selected execution mechanisms. There are also some other automatically generated methods which are used by `FOMConverterMgr` in runtime, developer can also modify these methods. Some snapshots of this file are given in Figure E-1 and Figure E-2.

```
....
/// User Defined Simple Data Types
typedef HLAfloat32LE Float;

/// User Defined Enumerated Data Types
public enum class HLAboolean { HLAfalse, HLAtrue };

/// User Defined Array Data Types
typedef array<HLAASCIIchar>^   HLAASCIIstring;

/// User Defined Fixed Record Data Types
public value struct Vec3f {
      Float X; Float Y; Float Z;

      /// This method will be used for any initialization
      void Initialize();

      /// This method will be used to get the size of this FRD
      static int GetSize();

      /// Used to get byte repres. of structure in a byte array
      array<Byte>^ GetBytes();

      /// Used to set corresponding value from byte array
      void SetBytes(array<Byte>^ iArg, int iStartIndex);
};
...
...
```

Figure E-1 Sample code that illustrate Data Types from "ConverterLib.h"

```
public ref class
FOMConverterMgr:RTIDotNetLibrary1516::FOMConverterMgrBase
{
      protected:
      FOMConverterMgr();

      ///@Begin Internal Methods!
      /// Auto generated Conversion Method prep. are done here
      virtual void UpdateMethod(unsigned int iConverterID,
              Dictionary<String^, array<Byte>^>^
iConverterInput,
              Dictionary<String^, array<Byte>^>^
      iConverterOutput ) override;

      virtual void ReflectMethod(unsigned int iConverterID,
              Dictionary<String^, array<Byte>^>^
      iConverterInput,
              Dictionary<String^, array<Byte>^>^
      iConverterOutput ) override;
      ...
      ///@End Internal Methods!
      ...
      /// Conversion methods used for publish directed Calls
      Void TemperatureConverterPublishDirectedFunc(
       Float% InputAttributeConnection,
       Float% OutputAttributeConnection);

      /// Conversion method used for subscribe directed HLA
Calls
      void TemperatureConverterSubscribeDirectedFunc(
       Float% OutputAttributeConnection,
       Float% InputAttributeConnection);
      ...
}
```

Figure E-2 Sample code that illustrate FOMConverterMgr from "ConverterLib.h"

o **ConverterLib.cpp**

This file contains the empty body blocks of `Conversion Methods` whose prototypes are generated in "ConverterLib.h" file. These methods with their provided input/output arguments name and their modeled data types are provided to developer who will fill these methods and use the provided arguments in her conversion operations. These conversion operations can range from a simple operation to complicated conversion calculations that make use of 3rd libraries. A snapshot of this file is given in Figure E-3.

```
...
/// Conversion method used for publish directed HLA Calls
void FOMConverterMgr::TemperatureConverterPublishDirectedFunc(
Float% InputAttributeConnection,Float% OutputAttributeConnection)
{
      /// Write your conversion code here
      OutputAttributeConnection=InputAttributeConnection*1.8f+32;
}

/// Conversion method used for subscribe directed HLA Calls
void FOMConverterMgr::TemperatureConverterSubscribeDirectedFunc(
Float% OutputAttributeConnection,Float% InputAttributeConnection)
{
      /// Write your conversion code here
      InputAttributeConnection=(OutputAttributeConnection-
            32)/1.8f;
}
...
```

Figure E-3 Sample code that illustrate Conversion Methods from "ConverterLib.cpp"

- **ConverterTypeLib.cpp**

This file contains implementation of data type support methods whose declarations are generated in "ConverterLib.h". Some snapshots of this file are given in Figure E-4 and Figure E-5.

```
int Vec3f::GetSize() {
      /// Calculate the size by adding all sizes of members
      return sizeof( Float) + sizeof( Float) + sizeof( Float);
}

array<Byte>^ Vec3f::GetBytes() {
      /// Allocate data for bytes
      array<Byte>^ result = gcnew array<Byte>( GetSize() );

      int currentIndex = 0;
      /// Get byte representation of all members and copy them
      /// into result array.
      Array::Copy( BitConverter::GetBytes( Z), 0, result,
            currentIndex, sizeof( Z) );
      currentIndex += sizeof( Z);
      Array::Copy( BitConverter::GetBytes( Y), 0, result,
            currentIndex, sizeof( Y) );
      currentIndex += sizeof( Y);
      Array::Copy( BitConverter::GetBytes( X), 0, result,
            currentIndex, sizeof( X) );
      currentIndex += sizeof( X);

      ///Return byte representation of whole structure
      return result;
}
```

Figure E-4 Sample code that illustrate GetSize/GetBytes Methods from "ConverterTypeLib.cpp"

```
void Vec3f::SetBytes(array<Byte>^ iArg, int iStartIndex){
      int currentDstIndex = iStartIndex;
      /// Put byte representation of all members
      /// into given array.
      SetValue(X, iArg, currentDstIndex);
      currentDstIndex += sizeof( HLAfloat32LE);
      SetValue(Y, iArg, currentDstIndex);
      currentDstIndex += sizeof( HLAfloat32LE);
      SetValue(Z, iArg, currentDstIndex);
      currentDstIndex += sizeof( HLAfloat32LE);
}
```

Figure E-5 Sample code that illustrate SetBytes Method from "ConverterLib.cpp"

- **FOMConverterMgr.cpp**

These file contains the implementation of `FOMConveterMgr` methods like class constructor and internal service processing methods. As stated before, for custom

behavior, these methods can be modified. A snapshot of this file is given in Figure E-6.

```
void FOMConverterMgr::UpdateMethod(unsigned int iConverterID,
      Dictionary<String^, array<Byte>^>^ iConverterInput,
      Dictionary<String^, array<Byte>^>^ iConverterOutput)
{     switch(iConverterID)
      {...
            /// When a temperature converter related member is
            /// updated call necessary Conversion Method
            case TemperatureConverterID:
            {
                  /// Prepare inputs/outputs for Conversion
                  /// Method
                  Float inputArg0;
                  SetValue( inputArg0,
                        iConverterInput["AverageTemperature"],
                        0);

                  Float outputArg0;

                  /// Call Conversion Method
                  TemperatureConverterPublishDirectedFunc(
                        inputArg0, outputArg0 );

                  /// Convert outputs to Byte array
                  iConverterOutput["Temperature"] =
                        BitConverter::GetBytes( outputArg0);
            }
            break;
      } ...
} ...
```

Figure E-6 Sample code that illustrate UpdateMethod from "FOMConverterMgr.cpp"

o **AttributeConverterIDs.h, ParameterConverterIDs.h**

These file contains identifiers that assigned for each of `AttributeConverter` and `InteractionConverter` created in modeling phase for internal purposes. A snapshot of "AttributeConverterIDs.h" is given in Figure E-7.

```
...
const unsigned int CoordinateConverterID = 0;
const unsigned int WindDirectionConverterID = 1;
const unsigned int WindSpeedConverterID = 2;
const unsigned int TemperatureConverterID = 3;

...
```

Figure E-7 Sample code from “AttributeConverterIDs.h”

○ **ConverterLibCommon.h, ConverterLibCommon.cpp**

These files contain supporting function declaration and implementations for developer to convert given byte array to Simple Data Types. Some snapshots of “ConverterLibCommon.h” and “ConverterLibCommon.cpp” files are given in Figure E-8 and Figure E-9.

```
...

/// Convert given byte array to Int32
extern void SetValue(System::Int32% iArg, array<Byte>^ iSrc,
            int iIndex);

/// Convert given byte array to Int64
extern void SetValue(System::Int64% iArg, array<Byte>^ iSrc,
            int iIndex);

/// Convert given byte array to Float
extern void SetValue(System::Single% iArg,
            array<Byte>^ iSrc, int iIndex);

/// Convert given byte array to Double
extern void SetValue(System::Double% iArg,
            array<Byte>^ iSrc, int iIndex);

...
```

Figure E-8 Sample code from “ConverterLibCommon.h”

```
/// Convert given byte array to Int32
void SetValue(System::Int32% iArg, array<Byte>^ iSrc,
            int iIndex)
{
      iArg = BitConverter::ToInt32(iSrc, iIndex);
}

/// Convert given byte array to Int64
void SetValue(System::Int64% iArg, array<Byte>^ iSrc,
            int iIndex)
{
      iArg = BitConverter::ToInt64(iSrc, iIndex);
}

/// Convert given byte array to Double
void SetValue(System::Double% iArg, array<Byte>^ iSrc,
            int iIndex)
{
      iArg = BitConverter::ToDouble(iSrc, iIndex);
}

/// Convert given byte array to Single
void SetValue(System::Single% iArg, array<Byte>^ iSrc,
      int iIndex)
{
      iArg = BitConverter::ToSingle(iSrc, iIndex);
}
```

Figure E-9 Sample code from “ConverterLibCommon.cpp”

- **Other files**

The interpreter also generates MS Visual Studio .NET 2005 project and other necessary files for developers who have this tool with default settings needed for compilation and makefile templates for developer who have only command line tools.