DENİZ ÇETİNKAYA METU 2005

A METAMODEL FOR THE HIGH LEVEL ARCHITECTURE OBJECT MODEL

DENİZ ÇETİNKAYA

AUGUST 2005

A METAMODEL FOR THE HIGH LEVEL ARCHITECTURE OBJECT MODEL

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

BY

DENİZ ÇETİNKAYA

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

AUGUST 2005

Approval of the Graduate School of Natural and Applied Sciences.

_____
Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____
Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____
Asst. Prof. Dr. Halit Oğuztüzün
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Ahmet Coşar (METU,CEng) _____

Asst. Prof. Dr. Halit Oğuztüzün (METU,CEng) _____

Assoc. Prof. Dr. Veysi İşler (METU,CEng) _____

Asst. Prof. Dr. Kayhan İmre (HACETTEPE,CEng) _____

Dr. Cevat Şener (METU,CEng) _____

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 : Deniz Çetinkaya

Signature :

ABSTRACT

A METAMODEL FOR
THE HIGH LEVEL ARCHITECTURE OBJECT MODEL

Çetinkaya, Deniz
M.S., Department of Computer Engineering
Supervisor: Asst. Prof. Dr. Halit Oğuztüzün

August 2005, 69 pages

The High Level Architecture (HLA), IEEE Std. 1516-2000, provides a general framework for distributed modeling and simulation applications, called federations. HLA focuses on interconnection of interacting simulations, called federates, with special emphasis on reusability and interoperability. An HLA object model, be it a simulation object model (SOM), a federation object model (FOM) or the management object model (MOM), describes the data exchanged during federation execution. This thesis introduces a metamodel for the HLA Object Model, fully accounting for IEEE Std. 1516.2. The metamodel is constructed with GME (Generic Modeling Environment), a meta-programmable tool for domain-specific modeling, developed at Vanderbilt University. GME generates a design environment for HLA object models having the HLA OM metamodel as input. This work can be regarded as a step for bringing model-integrated computing to bear on HLA-based distributed simulation.

Keywords: High Level Architecture, Object Model Template, metamodeling, Generic Modeling Environment

ÖZ

YÜKSEK SEVİYE MİMARİSİ NESNE MODELİ
İÇİN BİR METAMODEL

Çetinkaya, Deniz
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Asst. Prof. Dr. Halit Oğuztüzün

Ağustos 2005, 69 sayfa

IEEE Std. 1516 ile tanımlanan Yüksek Seviye Mimarisi (YSM), Modelleme ve Simülasyon (M&S) uygulamaları için genel bir altyapı tanımlamaktadır. YSM simulasyonların yani federelerin birbirleriyle olan etkileşimleri üzerinde durur ve tekrar kullanılabilirlik ile birlikte çalışabilirlik ilkelerine özel önem verir. YSM nesne modelleri, simulasyon nesne modeli (SOM), federasyon nesne modeli (FOM) ve federasyon yönetimi nesne modeli (MOM) olarak tanımlanır ve federasyon hedeflerini gerçekleştirmek için kullanılan bilgiyi tanımlarlar. Bu tezde YSM nesne modelleri için IEEE 1516.2 standardını tamamen kapsayan bir metamodel önerilmiştir. Bu metamodel meta programlanabilir bir modelleme aracı olan ve Vanderbilt Üniversitesi tarafından geliştirilen GME (Generic Modeling Environment) aracı ile tanımlanmıştır. GME tanımlanan metamodeli kullanarak YSM nesne modelleri için bir modelleme ortamı oluşturur. Bu çalışma bütünleşik gelişim sürecinin (MIC) YSM'ye uygulanması konusunda bir basamak olarak görülebilir.

Anahtar Kelimeler: Yüksek Seviye Mimarisi, Nesne Modeli, metamodelleme, Generic Modelleme Ortamı

To my Daughter

ACKNOWLEDGMENTS

The author wishes to thank her supervisor Asst. Prof. Dr. Halit Oğuztüzün for his guidance, advice, criticism, encouragements and insight throughout this research.

The author would also like to thank Prof. Dr. Müslim Bozyiğit for giving her the opportunity to work with him.

The author also wishes to gratefully acknowledge Ph.D. students Gürkan Özhan, Mehmet Adak and Okan Topçu for their comments on improving the quality of the thesis.

The author also wishes to thank her mother Gülşen Küçükkeçeci, her father Muhsin Küçükkeçeci, her brothers Cihan and Onur for their support during the preparation of this thesis.

At last, but the most, her special thanks are due to her husband Orhan Çetinkaya for his endless patience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DIF	Data Interchange Format
DMSO	Defense Modeling and Simulation Office
DoD	The United States Department of Defense
DSM	Domain Specific Modeling
DSME	Domain-Specific MIPS Environment
DSML	Domain Specific Modeling Language
FDM	Federation Design Model
FEDEP	Federation Development and Execution Process model
FOM	Federation Object Model
GME	Generic Modeling Environment
HLA	High Level Architecture
IEEE	Institute of Electrical and Electronic Engineers
MDA	Model Driven Architecture
MIC	Model Integrated Computing
MIPS	Model Integrated Program Synthesis
MOF	Meta Object Facility
MOM	Management Object Model
OCL	Object Constraint Language
OM	Object Model
OMDT	Object Model Development Toolkit
OMG	Object Management Group
OMT	Object Model Template
RTI	Run Time Infrastructure
SOM	Simulation Object Model
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

CHAPTER 1

INTRODUCTION

The High Level Architecture (HLA) is a key technology in Modeling and Simulation (M&S) area for distributed simulations. Zeigler and his colleagues [7] described the computer simulations development process in terms of three major elements: the source system, the model, and the simulator. Modeling is primarily concerned with the relationships between source systems and models, while simulation refers to relationships between simulators and models. The increasing interest in modeling and simulation has also increased the importance of the HLA.

Currently the HLA is an IEEE (Institute of Electrical and Electronic Engineers) standard as the IEEE Std. 1516 specification, which is also our basic reference [22]. This standard defines the HLA as an integrated architecture that provides a general framework within which simulation developers can structure and describe their simulation applications [9].

In order to perform simulation, the HLA defines some basic terms: federation, federate, federation execution, run time infrastructure and others [4]. The combined simulation system created from the constituent simulations is a federation; each simulation that is combined to form a federation is a federate; and a session of a federation executing together is a federation execution. A federation contains a supporting software called the Run Time Infrastructure (RTI), a number of federates and a common object model for the data exchanged between these federates. The HLA gives special attention to interoperability and reusability. It has three main components: HLA Rules [9], Object Model Template (OMT) [11] and Federate Interface Specification [10]. Rules are a set of rules that apply to HLA federations and federate. OMT describes the data used by a particular model called object model and addresses reusability. Federate Interface Specification describes a generic communications interface that allows simulation models to be connected and coordinated. It also addresses interoperability issues.

HLA object models are composed of a group of interrelated components. IEEE Std. 1516.2 [11] gives motivation for new representation techniques for object models saying that:

> "The information content of components can be represented in many different ways or presentations. A presentation is the formatting of the information contained in the object model in a particular manner for a particular purpose. All HLA object models shall be capable of being presented in both the OMT tabular format and the OMT Data Interchange Format (DIF) format."

OMT tabular format uses tables to save the information in the object model, where OMT DIF format expresses object models with nested tags similar to XML (eXtensible Markup Language) [26]. Both of them are well defined and text based presentations. Unfortunately, at design steps textual presentations may lack on understandability, reusability and upgradeability. Defining a visual layer on top of OMT to provide a visual representation, can help easier understanding for complex federations and can reduce the design time with better reusability. Many researchers proposed various representations for object models as well as for other federation design issues. Next section introduces some related work. In this thesis, we have adopted metamodeling, which is another representation technique that has been widespread recently, where metamodel is a definition of an architectural language in the form of a model. Section 3.2 and 3.3 discusses the reasons of choosing the metamodeling approach.

Though we want to define a visual layer with metamodeling approach, we searched for a domain specific modeling environment, which will give us the opportunity to describe our metamodel and provide support for further design issues. The Generic Modeling Environment (GME), developed at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University, is a configurable toolkit that supports the easy creation of domain-specific modeling and program synthesis environments [39]. It is free and open software. It is actively supported by ISIS. So we have chosen GME as the modeling environment.

This thesis proposes a metamodel for HLA Object Model (OM) and describes this metamodel in a domain specific modeling environment, which is GME, in order to define a visual representation. In GME, metamodels are defined in modeling paradigms using GME metamodel. After describing the HLA OM modeling paradigm, the GME modeling

environment creates a design environment for domain models by using this proposed metamodel automatically. Then the generated design environment can be used to design HLA object models.

1.1 Related Work

Object Model Tools are expected to have facilities on designing, developing, managing and sharing object models; by providing object model development tools, object model libraries or object model data dictionaries [4]. Many researchers worked on these subjects, in this section we point out some studies that we have examined. In our view an object model tool should be based on a metamodel. Presumably, each of these tools has a native representation scheme for object models. We believe that tool integration and tool extensibility will be greatly facilitated by the use of metamodels that are accessible to users. Adoption of standardized metamodels will further improve integration and extensibility issues.

1.1.1 An Integrated Tool for HLA

This research project at Computer Engineering Department of Hacettepe University focuses on modeling, code generation, monitoring and testing issues for HLA [29]. Researchers aim to develop an integrated toolset that will improve and accelerate federation development and testing process. Their work is based on UML extensions.

1.1.2 Object Model Development Toolkit

Object Model Development Toolkit (OMDT) by AEgis Technologies [20] is a tool which provides an environment for creating, editing, manipulating, and managing HLA object models with an easy to use interface.

1.1.3 Calytrix SIMplicity

Calytrix SIMplicity is a visual tool for developing distributed HLA simulations from new and pre-existing components (federates) [41]. It provides an integrated development environment to design and manage object models, to design federates, to create complex mappings with third-party federates, to automatically generate code, and more. Calytrix Technologies (Australia) is a leading company in simulation area and Calytrix

SIMplicity [25] is listed as a committed product, which means this product is a valuable resource in its application area, in Object Management Group (OMG) web site [23].

1.1.4 Visual OMT

Visual OMT is a graphical object model development tool for efficient development and maintenance of HLA Object Models [19]. It provides an integrated environment in which the HLA object models of any simulation or federation can be loaded and edited [34]. It is a product of Pitch Company (Sweden) [24], which supplies commercial tools and services to the modeling and simulation community.

1.1.5 GENESIS

GENESIS project of the French National Air and Space Agency (ONERA) aims to develop a tool for the design and development of HLA simulations [28]. It is financed by the French General Armament Directorate (DGA). With the current version of GENESIS defining object models, C++ code and configuration file generation are provided. It started in January 2003 and runs for 3 years.

1.1.6 A Framework for Ontology Based Federation Development

Ontology can be defined as an explicit specification of a conceptualization in [33]. A research study held in Systems Realization Laboratory at Georgia Institute of Technology, aims to define a framework for ontology based federation development. The major components of this framework are the federate ontologies, target federation ontology and a metamodel that corresponds to the OMT, called the World Ontology [42].

1.2 Thesis Overview

After this introduction chapter, the thesis continues with Chapter 2 which introduces HLA and IEEE Std. 1516. Metamodeling concepts and the GME Metamodel are explained in Chapter 3. Metamodel for High Level Architecture Object Model is proposed and discussed in Chapter 4. Finally, conclusions are drawn and future work is suggested in Chapter 5. The appendices introduce complementary material: Appendix A contains a User's Guide for designing object models and Appendix B contains a sample output file.

CHAPTER 2

HIGH LEVEL ARCHITECTURE (HLA)

IEEE standard defines the HLA as an integrated architecture that provides a general framework within which simulation developers can structure and describe their simulation applications. The HLA is not software, it provides a common framework. The HLA was first developed by the Defense Modeling and Simulation Office (DMSO) [21] of The United States Department of Defense (DoD), in order to support reuse and interoperability across the large numbers of different types of simulations and so reduce the cost of the projects [2]. The HLA Baseline Definition was approved by the Under Secretary of Defense for Acquisition and Technology (USD (A&T)) in 1996. After this approval the HLA is proposed as the standard technical architecture for all DoD simulations.

OMG [23] adopted the 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. The HLA was approved as an open standard through the IEEE, namely IEEE Standard 1516, in September 2000. In November 2000 the HLA is identified as the preferred architecture within the DoD [21].

Officially designated as IEEE Std. 1516-2000 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA), the standard was prepared by the HLA Working Group, sponsored by the Simulation Interoperability Standards Committee (SISC) of the IEEE Computer Society. The standard consists of four related standards:

- IEEE Std. 1516-2000: IEEE Standard for M&S HLA Framework and Rules [9]
- IEEE Std. 1516.1-2000: IEEE Standard for M&S HLA Federate Interface Specification [10]
- IEEE Std. 1516.2-2000: IEEE Standard for M&S HLA Object Model Template (OMT) Specification [11]

- IEEE Std. 1516.3: IEEE Recommended Practice for HLA Federation Development and Execution Process (FEDEP) [12]

Next sections give brief overviews of these standards.

2.1 Framework and Rules

This is the main standard which provides an overview of the HLA and defines a set of rules that apply to HLA federations and federates [9]. Simulation Object Model (SOM) and Federation Object Model (FOM) mentioned here are explained in section 2.3. The rules for federations are as follows:

- Federations shall have an HLA FOM, documented in accordance with the HLA OMT.
- In a federation, all simulation-associated object instance representation shall be in federates, not in the RTI.
- During a federation execution, all exchange of FOM data among joined federates shall occur via the RTI.
- During a federation execution, joined federates shall interact with the RTI in accordance with the HLA federate interface specification.
- During a federation execution, an instance attribute shall be owned by at most one joined federate at any given time.

The rules for federates are as follows:

- Federates shall have an HLA SOM, documented in accordance with the HLA OMT.
- Federates shall be able to update and/or reflect any instance attributes and send and/or receive interactions, as specified in their SOMs.
- Federates shall be able to transfer and/or accept ownership of instance attributes dynamically during a federation execution, as specified in their SOMs.
- Federates shall be able to vary the conditions under which they provide updates of instance attributes, as specified in their SOMs.

- 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.

2.2 Federate Interface Specification

This specification defines the standard services and interfaces to be used by federates to be able to interact other federates in a distributed federation execution. Federate interface specification extends the HLA Rules and discusses the HLA concepts in detail.

As stated before HLA federates interact with an underlying runtime infrastructure, which is RTI. Use of RTI software is required for HLA to coordinate the operations and data exchange during a federation execution [10]. The HLA Federate Interface Specification defines the services that the RTI software provides. Figure 2.1 shows the high level logical view of an HLA federation in order to depict the communication between federates and RTI.

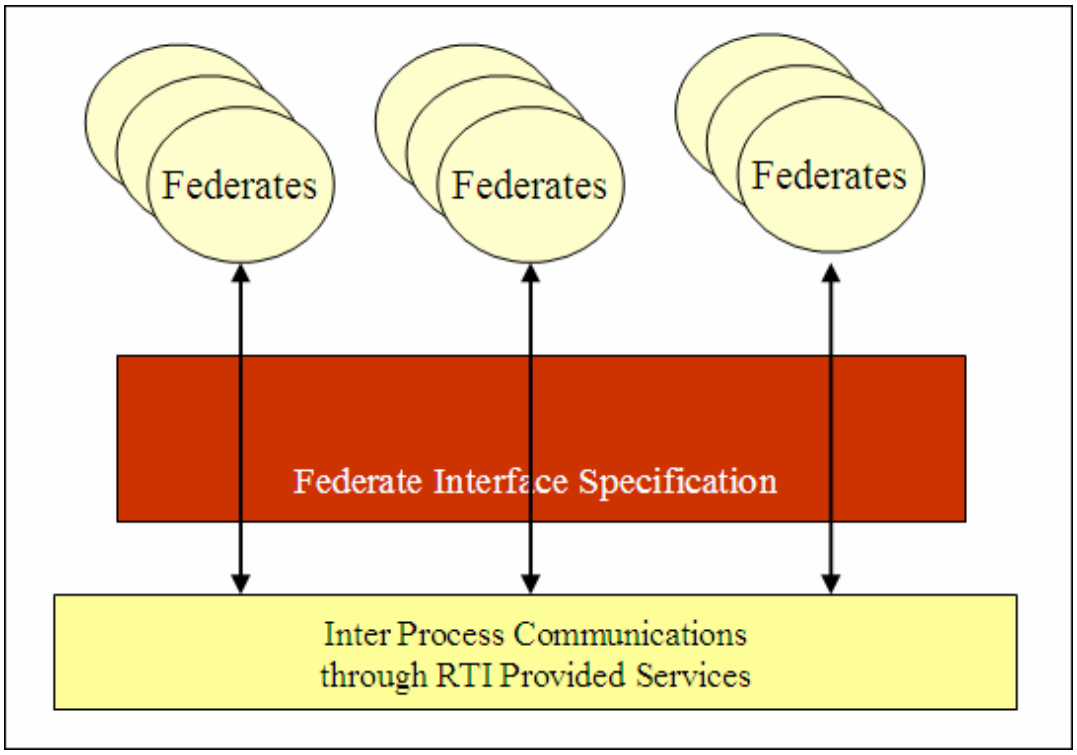

Figure 2.1: Components of an HLA federation [2].

The services defined in this specification are divided into 7 service groups. These are outlined as the follows:

- **Federation management**: Creating, modifying, deleting and dynamic control of a federation execution are provided by federation management services. This group includes join to a federation execution and resign from a federation execution services of federates; and some federation synchronization point services.
- **Declaration management**: Joined federates use the services in this group to declare their interest to an object class attribute or an interaction class. Also these services are used to declare intention to generate information. Namely publish/unpublish and subscribe/unsubscribe services are included.
- **Object management**: The services in this group deal with the registration, modification, and deletion of object instances and the sending and receiving of interactions. Also this group includes some transportation type services.
- **Ownership management**: The services in this group are used to transfer ownership of instance attributes among joined federates.
- **Time management**: Messages sent by different joined federates are delivered in a consistent order throughout the federation execution by the time management services and associated mechanisms.
- **Data distribution management**: The services in this group provide information on data relevance at different levels and allow refining the data requirements. These services may be used to reduce the transmission of irrelevant data in terms of bounds, regions or sets.
- **Support services**: This group includes miscellaneous services for performing such actions as setting advisory switches, manipulating regions, or RTI start-up and shutdown.

The support services specification also introduces the Management Object Model (MOM). MOM is defined in this specification instead of OMT specification, because MOM provides facilities for access to RTI operating information during federation execution, which can be used by joined federates, where a joined federate is a member of

a federation execution. Section 4.2.2 gives more information about MOM and how it is modeled in HLA OM Metamodel.

2.3 Object Model Template (OMT) Specification

OMT introduces two types of object models: an HLA Simulation Object Model (SOM) which describes an individual federate and a Federation Object Model (FOM) which describes a federation. HLA object models identify the data exchanged at runtime to achieve federation objectives. The OMT Specification defines the format and syntax of HLA object models.

Firstly, the standard presents object models in OMT tabular format. The OMT consists of 14 components that are presented as tables:

- **Object model identification table**: This component identifies 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**: If we understand attributes as objects with certain characteristics, then object class is a collection of attributes. Object class structure table includes all federate or federation object classes and their class-subclass relationships.
- **Interaction class structure table**: An explicit action taken by a federate that may have some effect or impact on another federate within a federation execution is called an interaction. Interaction class structure table includes all federate or federation interaction classes and their class-subclass relationships.
- **Attribute table**: Each object class may have attributes. This table includes features of object attributes in a federate or federation.
- **Parameter table**: Each interaction class may have parameters. This table includes features of interaction parameters in a federate or federation.
- **Dimension table**: Each attribute or interaction that uses data distribution management services has a set of available dimensions. This component specifies dimensions for filtering instance attributes and interactions.

- **Time representation table**: This component defines the usage of time stamps and lookahead characteristics of both federates and federations.
- **User-supplied tag table**: Federates can supply tags with certain HLA services to provide additional coordination and control. This table defines these tags.
- **Synchronization table**: This component specifies the representation and data types used in HLA synchronization services.
- **Transportation type table**: It describes mechanisms used for the transportation of data.
- **Switches table**: It includes the 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 descriptive information can be added to any element of the object model. This component includes these notes.
- **FOM/SOM lexicon**: It includes verbal definitions for the objects, attributes, interactions, and parameters used in the HLA object model.

After describing these components in detail, the standard introduces OMT DIF format. The HLA OMT DIF is a standard data exchange format, which is built on a common metamodel that represents the information needed to represent and manage HLA object models.

2.4 Recommended Practice for HLA FEDEP

Guidance in the process of developing HLA federations is provided by the Federation Development and Execution Process model. FEDEP defines how HLA applications are constructed and executed, though it does not prescribe a single way of constructing an HLA federation. In other words the actual process used to develop and execute HLA federations could vary within or across different user applications. However, at an abstract level, it is possible to identify a sequence of seven basic steps that HLA

federation developers should follow to develop and execute their federations. Figure 2.2 shows the seven step FEDEP model.

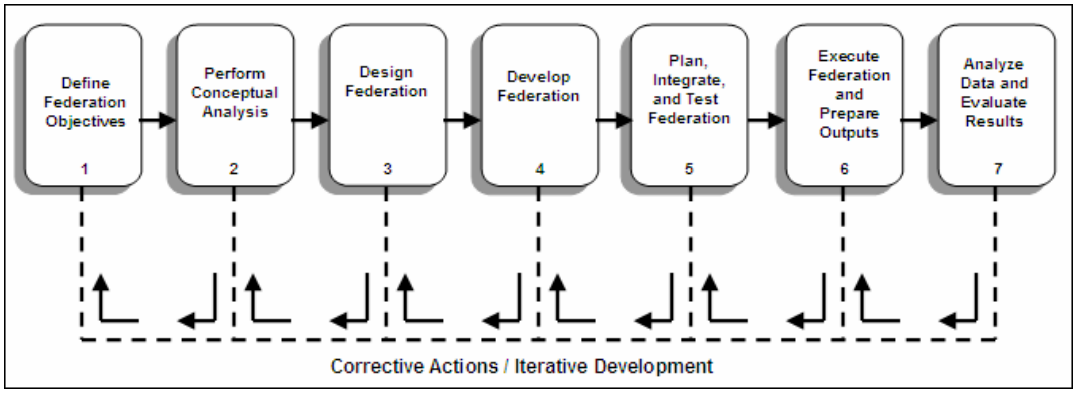

Figure 2.2: Top-level view of FEDEP model [12].

The FEDEP model identifies these steps in detail and defines a process for guiding the development of an HLA federation. More information about High Level Architecture can be found in [4], [21], [10], [11].

CHAPTER 3

GENERIC MODELING ENVIRONMENT (GME)

This chapter provides description of the modeling environment GME. We have used GME version 4.11.10 in this research. Defining a metamodel for HLA OM is an instance of domain specific modeling. In this context we can talk about Model Driven Architecture (MDA) and more specifically Domain Specific (DS) MDA.

DS MDA, also known as Model Integrated Computing (MIC), is touted as an effective and efficient method for developing, maintaining, and evolving large-scale, domain-specific software [40]. Karsai and his colleagues in [35] and [36] introduce tools for MIC, shown in Figure 3.1. In order to set up a specific MIC process first the metamodels for the domain, the target, and the transformations should be created. Then a domain specific model editor and a model transformation tool can be built with meta-level tools. The model editor is then used to create and modify domain models, while the transformation tool is used to convert the models into target models. Next section briefly reviews the MIC concept.

The GME is a meta-programmable modeling tool that supports the creation of domain-specific modeling and program synthesis environments [39]. In [8], the definition is extended and GME is described as a configurable model-integrated program synthesis tool, where configurable means that it can be programmed to work with vastly different domains. As Figure 3.1 indicates, a meta-programmable tool creates domain models after configuration of a domain metamodel. Section 3.4 describes the core modeling concepts supported by GME.

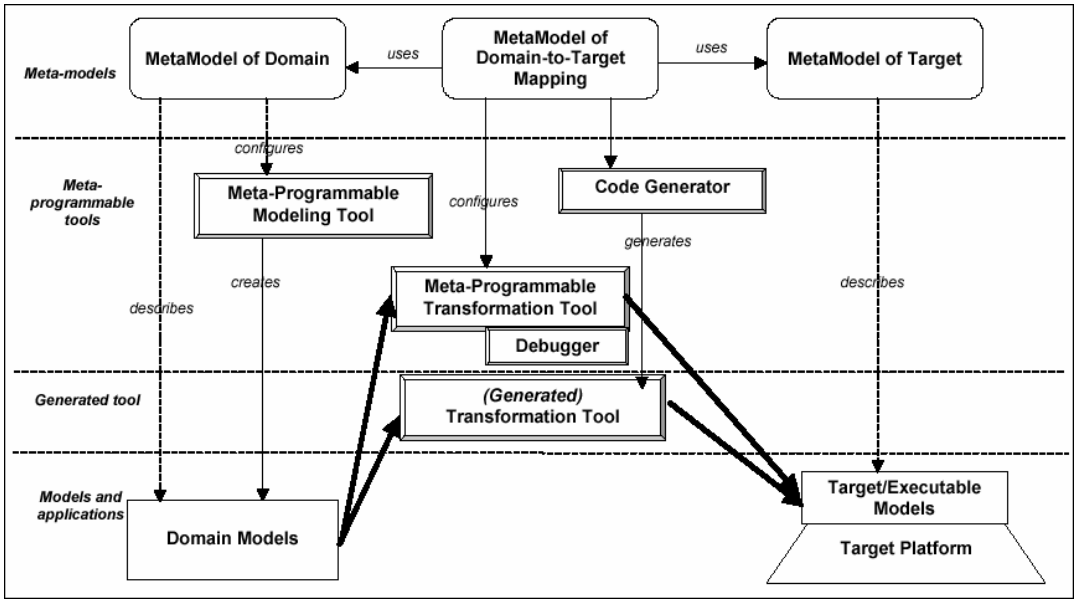

Figure 3.1: Tools of MIC [35].

There are some characteristics of the GME that make it a valuable tool for the construction of domain-specific modeling environments.

- The GME provides generic modeling primitives [38], which are specialized to create the domain-specific modeling concepts through meta-modeling.
- A metamodel for a domain is defined as the modeling paradigm described in the form of a formal modeling language, where the modeling paradigm is a set of requirements that specifies the ontology of the domain [40]. GME paradigms are generated from formal modeling environment specifications.
- The GME 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.
- Models are formed as graphical, multi-aspect, attributed entity-relationship diagrams [8]. The dynamic semantics of a model is determined later during the model interpretation process.
- It supports various techniques (hierarchy, multiple aspects, sets, references, and explicit constraints) for building large-scale, complex models.

- It contains some integrated model interpreters that perform translation and analysis of models and provides defining new interpreters.
- The GME supports multiple paradigms and enables meta-model composition [38].
- The GME is Windows based and well organized graphical user interfaces make it easy to use.
- The GME is an ongoing academic research project that one can use freely for non-commercial purposes. Its source codes are available.
- The GME contains an extended constraint manager which is fully compliant with the Object Constraint Language (OCL) 1.4 specification [16], which is a well known OMG specification [8].
- The GME-metamodel is implemented using UML 1.4 specification, which is also a well known OMG specification [8].

Besides the advantages, the GME needs some improvements and upgrades. The most important expectation is about OCL 1.4 and UML 1.4 specifications, which are not up-to-date. OCL 2.0 [18] and UML 2.0 [15] are already available and the academia waits for the upgrades. Also a meta-programmable modeling tool with Meta Object Facility (MOF) 2.0 is another expected issue. The facility for metamodeling with directly MOF 2.0 metamodel will enhance the GME. The good news is that the researchers at ISIS have started upgrading GME.

In this thesis, the proposed metamodel for HLA OM is a GME modeling paradigm based on the GME-metamodel. GME generates a design environment for HLA object models automatically using the proposed metamodel that is represented as a modeling paradigm.

In accordance to the objectives of this thesis, MIC, the metamodeling approach and the GME modeling concepts are presented in the following sections.

3.1 Model Integrated Computing (MIC)

Model integrated computing, or domain specific Model Driven Architecture (MDA), is a methodology for developing, maintaining, and evolving large-scale, domain-specific

software, which uses MDA concepts and metamodel approach [40]. The MDA is defined by the OMG and based on the importance of models in the software development process [13]. In MDA the formal models are the building stones of software development [3].

A group of entities that share the same characteristic or exhibit similar functionality is known as a domain [43]. Domain specific modeling is an approach to model a system with the terminology and concepts from a specific domain [32], where data structures and logic are abstracted beyond programming. Domain Specific Modeling Language (DSML) is a language specifically designed to represent and implement the particular problem domain concepts. MIC allows the synthesis of application programs from models by using customized domain-specific Model Integrated Program Synthesis (MIPS) environments. Once a domain-specific modeling language has been formally defined, a meta-level translation can be performed to synthesize the Domain-Specific MIPS Environment (DSME) [37] from the metamodel. The DSME is then used by domain experts to create various models of domain-specific systems. Once one or more domain models exists, model interpreters are used to perform semantic translations on the models in order to generate executable models or perform various types of data translation and analysis. Model transformations are the most important step in the MIC process, which are used to generate target domain specific models.

A MIPS environment operates according to a modeling paradigm [40]. A modeling paradigm is a set of requirements that governs how systems within the domain are to be modeled, i.e., it defines a language for modeling systems in the domain. The modeling paradigm is captured in the form of formal modeling language specifications called a metamodel. So, the MIC applies the metamodel based approach to domain specific applications. Figure 3.2 shows the MIC process summarized above.

We believe that applying domain specific MDA to HLA will contribute the distributed simulation design and development process in terms of formalism, reuse, verification, and validation.

3.2 Metamodeling

A model can be defined as an abstract representation of something so that we can view, manipulate, and reason about it in a well defined language; where a language defines

which elements can exist in the model [3]. If we define this well defined language with models, this mechanism is called metamodeling.

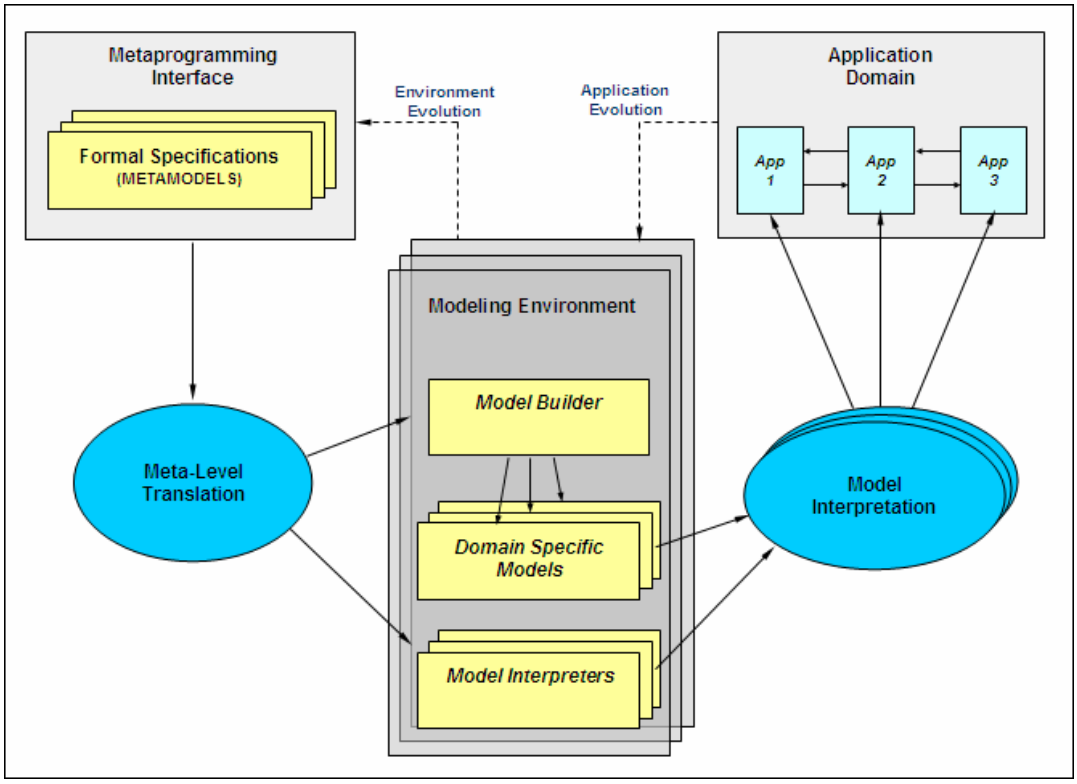

Figure 3.2: Overview of MIC process [37].

In fact, modeling and metamodeling are identical activities, the difference is the level of modeling. So it is also possible to mention meta-meta-models, which will describe the language to generate meta-models, and so on.

The OMG introduces a four-layer metamodeling infrastructure for defining modeling, metamodeling, and meta-metamodeling languages and activities in [17]. Table 3.1 describes each layer of this framework. Table 3.2 shows the metamodeling layers in our study mapped to OMG's four-layer framework. If we added a fifth layer, namely Meta-Meta-Metamodel, to this framework, which could be derived from the definition of Metamodeling, it would map to UML metamodel. The fifth layer UML Meta-Model describes the language to generate GME meta-models.

Table 3.1: Four-layer metamodeling infrastructure of OMG.

Layer	Definition
M3 Layer Meta Metamodels	This layer forms the foundation of the metamodeling hierarchy. It defines a language, namely meta-metamodeling language, for specifying a metamodel.
M2 Layer Metamodels	A metamodel is an instance of a meta-metamodel. This layer defines a language, namely metamodeling language, for specifying models.
M1 Layer Models	A model is an instance of a metamodel. This layer defines domain specific modeling languages that describe semantic domains.
M0 Layer Run-time instances	The metamodel hierarchy bottoms out at M0, which contains the run-time instances of model elements defined in a model.

Table 3.2: Mapping to layers of OMG.

Framework Model	Mapped Model
Meta Metamodel	GME Metamodel
Metamodel	HLA OM metamodel
Model	Domain model (e.g. Restaurant FOM)
Run-time instance	Domain model instance (e.g. object instances in a Restaurant federation execution)

Next section discusses the advantages of Metamodeling with a comparison with UML profiling mechanism, which is another domain specific modeling approach.

3.3 Metamodeling vs. UML Profiling Mechanism

The Unified Modeling Language is an industry standard language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. Research on domain specific modeling using UML has focused on the UML profiling mechanism [44] [45]. The specification of UML includes extension mechanisms that allow UML to be adapted to meet specific needs of a domain.

A UML profile is a stereotyped package that contains model elements that have been customized for a specific domain or purpose using extension mechanisms. A profile may specify model libraries on which it depends and the metamodel subset that it extends. In most cases a UML profile consists of a predefined set of stereotypes, tagged values and constraints. In [1], a stereotype is defined as the main extension mechanism providing a method to extend a meta-class. Tagged values are defined as properties attached to UML elements, and constraints are defined as the rules that restrict the semantics of one or more elements in UML.

After this brief introduction, we will compare metamodels to UML profiles by listing some differences, also other differences may occur.

- UML Profiles extend the UML syntax, while metamodels define entirely new modeling languages. In other words, profiling yields an extended UML for the intended domain, while metamodeling yields a specialized language, on the same footage as UML, for the domain.
- Stereotyping doesn't provide additional classes, while metamodels define their own model elements [30]. Furthermore, tool-specific restrictions may apply in this regard.
- UML Profiles may have some drawbacks when the domain metamodel and domain model both represented with the same notation. One may have problems with managing the models and the difference between layers may not have been come out.
- UML Profiles doesn't allow defining associations between stereotypes, but with metamodels you can define associations like inheritance, composition or others between model elements provided that the meta-metamodel includes their definition.

- UML Profiles doesn't allow changing the notation of any element; you can only define icons, while with metamodeling approaches you can define fundamentally new representations.

Lastly, in this thesis context we can say one more thing, which possibly makes metamodeling preferable. Firstly note that the HLA object model does not totally agree with the object model in the usual object oriented analysis and design (OOA&D) sense. Thus, a UML profile for HLA OM has to carry "extra luggage". This may hinder constraint specification and checking, and user input validation.

On the other hand, UML profiling allows one to stay with the same host UML tool, thus avoiding any tool switches between object-oriented design model (for the software) and object modeling (for the domain). The solution within metamodeling approach would be to invoke model transformations between the two modeling environments, and to utilize the XMI (XML Metadata Interchange) standard to carry the models across tools.

3.4 GME Architecture and Modeling Concepts

This section describes the core modeling concepts provided within the GME and their relationships. The GME supports various techniques for building large-scale complex models.

In GME the domain specific modeling languages are viewed as the modeling paradigms which contain all the syntactic, semantic, and presentation information regarding the domain. A modeling paradigm describes which concepts will be used to construct models, what relationships may exist among these concepts, how the concepts may be organized, and the rules governing the construction of models. The metamodels specifying the modeling paradigm are used to automatically generate the target domain-specific environment. The generated domain-specific environment is then used to build domain models, which can be used in different synthesis tools. This process is called model interpretation [39].

The GME architecture is explained in [31], here we only emphasize to modeling concepts shown in Figure 3.3. The modeling paradigm metamodel, also known as the GME metamodel introduces some concepts. In GME, Model is the basic concept that is an abstract object that represents something in the world and has state, identity, and

behavior. Models typically have parts, other objects contained within the model. Models can include models, atoms, FCOs (first class objects), references, sets, and connections. The purpose of the GME is to create and manipulate these models. Modeling paradigms may have several kinds of models.

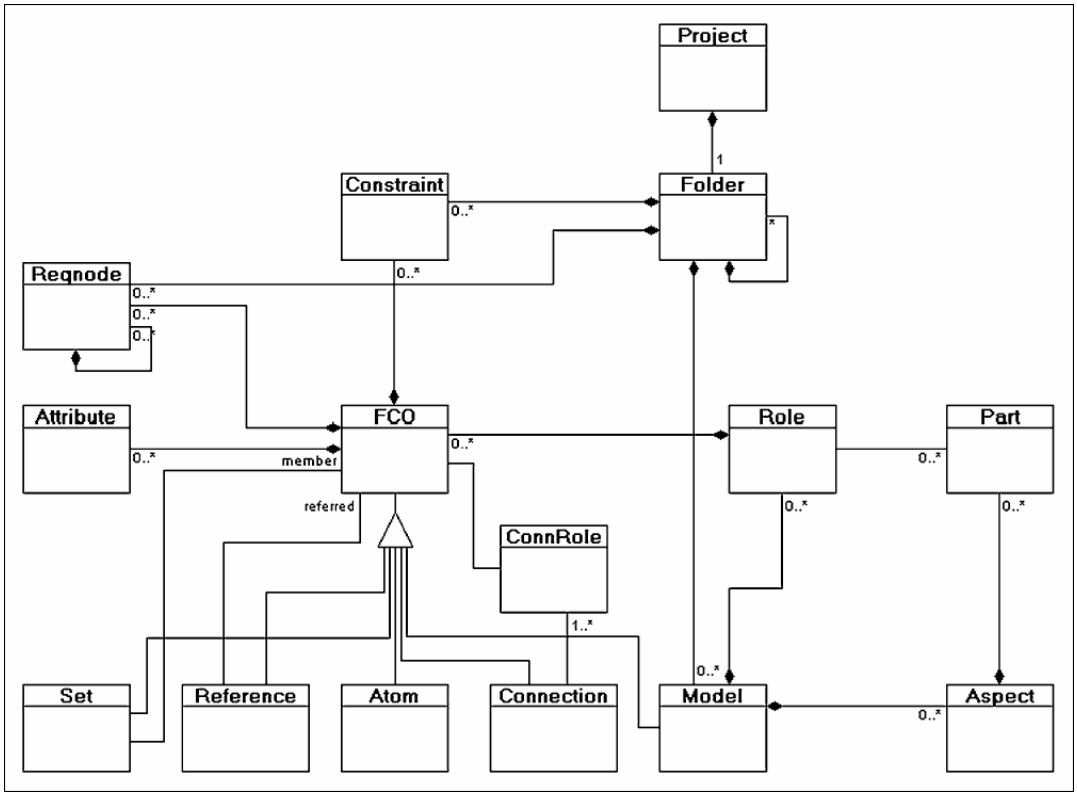

Figure 3.3: GME modeling concepts.

The modeling concepts of the GME are described below:

- **Project**: Project is the root container class.
- **Folder**: Folders are containers that help organize models.
- **FCO**: FCOs are first-class objects which must be abstract but can serve as the base type of any other elements.
- **Model**: Models are compound objects which can contain other model elements as stated above.

- **Atom**: Atoms are atomic objects which may not contain other model elements. They can have attributes
- **Set**: Sets can be used to specify a relationship among a group of objects. The only restriction is that all the members of a set must have the same parent and be visible in the same aspect.
- **Connection**: Connections are used to express a relationship between two objects.
- **Reference**: References are placeholders for FCOs (other than Connections) - similar to references in some programming languages.
- **Attribute**: Attributes are values of a simple type. Any FCO can have many attributes.
- **Aspect**: Aspects provide logical visibility partitioning to present different views of a model.
- **Constraint**: Constraints are Boolean expressions, which are evaluated over the object instances. The GME permits the specification of constraints using a variant of the OCL.

The GME modeling environment provides four views: classes, visualization, constraints and attributes. The classes view allows users (model builders) to define the basic class diagrams with models, references, atoms and other elements; the visualization view to define the aspects and to connect the modeling elements to the related aspects; the constraints view to define constraints and constraint functions with OCL and to connect the constraints to the modeling elements; the attributes view to define various attributes for the modeling elements. Figure 3.4 denotes sample views of GME.

We refer to GME User's Guide [8] for further details.

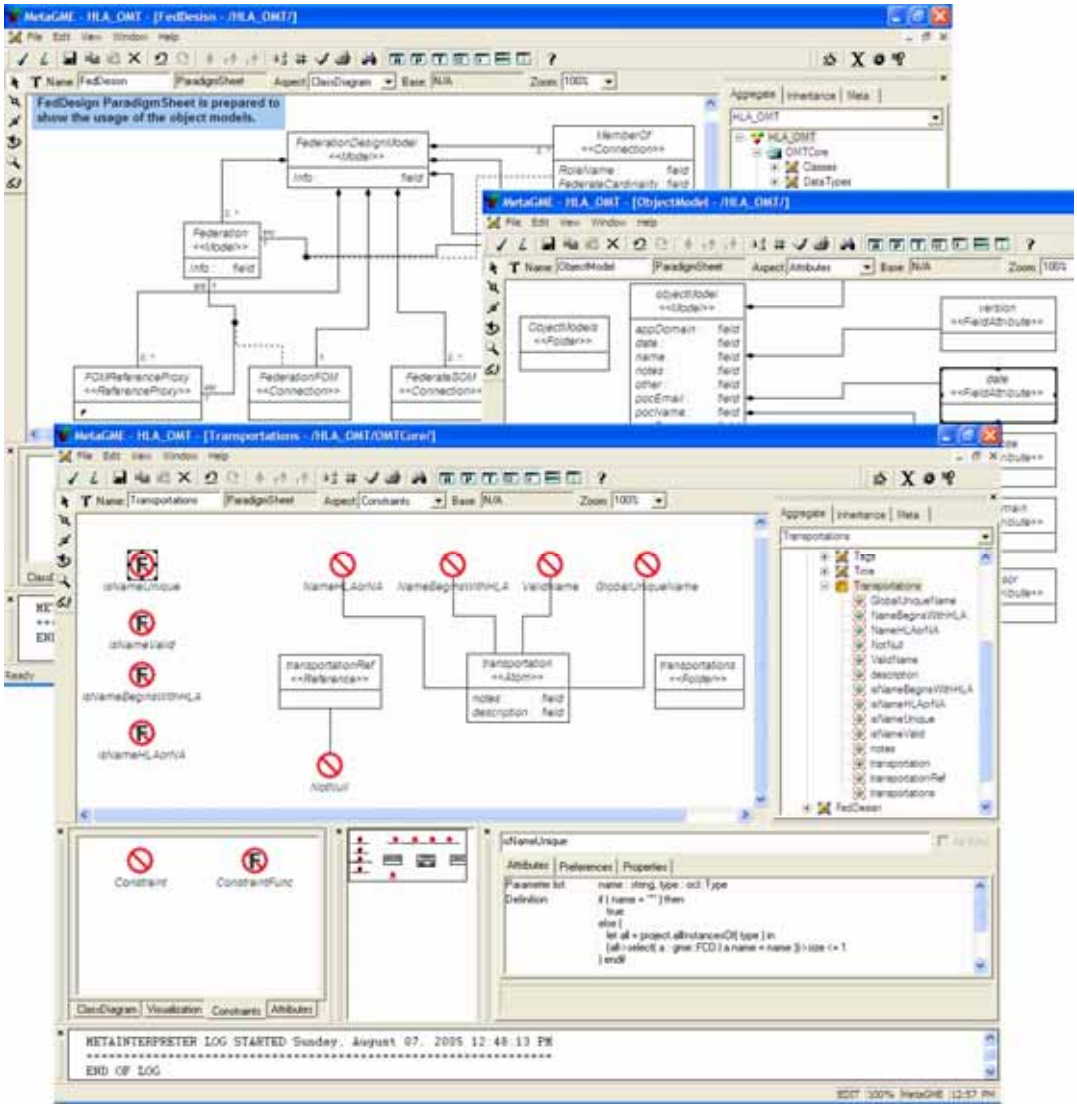

Figure 3.4: Sample views of GME.

CHAPTER 4

GME PARADIGM FOR THE HLA OBJECT MODEL

As we discussed in chapter 3, a modeling paradigm should be defined in GME in order to represent a metamodel which will constitute a domain specific modeling language. Then GME interprets this paradigm and generates a design tool which supports this modeling language. In our study we have modeled the HLA Object Model Template paradigm with GME.

This chapter presents the HLA OMT paradigm in an organized way. The class diagrams, aspects, attributes and constraints are explained and the rationale is discussed for each model. The metamodel introduced here is fully compliant with IEEE Std. 1516.2 and the modeled architecture refers to this standard in every detail. The modeling elements in our metamodel are named consistently with OMT Data Interchange Format (DIF). The metamodel is compatible with the HLA OMT tabular format and every metamodel element map to the related table column in OMT tabular format. Together with the attached comments, we regard the HLA OM metamodel as an alternative rendering of the HLA OMT standard IEEE 1516.2.

After introducing the metamodel, this chapter also shows some IEEE related default entries and an example. HLA OMT specification defines some default datatypes and transportation types; we included them in our metamodel as explained in section 4.2.1. Meanwhile, HLA Federate Interface Specification defines MOM, which must be included by all FOMs. We also modeled the MOM as explained in section 4.2.2. After preparing default libraries we have built a sample federation design model, based on the sample in HLA OMT specification, with our proposed metamodel; this is explained in section 4.2.3.

We also provide a User's Guide in Appendix A, which explains how to make HLA federation design model and object models with the design environment of the HLA OM paradigm. The intended user of our metamodel is a federate or federation developer.

4.1 HLA OM Metamodel as GME Paradigm

As already mentioned we have modeled the HLA OMT paradigm with GME. The proposed HLA OMT paradigm includes the Object Model paradigm sheet, the Federation Design paradigm sheet and the OMT Core folder. Paradigm sheets are separate models. OMT Core folder includes the definitions for classes, data types, dimensions, normalization functions, notes, switches, synchronization points, user supplied tags, time representations and transportations in separate paradigm sheets. Although we have focused on a metamodel for HLA OM, we have also defined a simple Federation Design metamodel for illustrative purposes. Object Models and OMT Core could be used individually if needed.

There are some rules which apply to the whole metamodel. One of them is the naming constraint: Names in object models can be constructed from a combination of letters, digits, hyphens, and underscores with no spaces or other breaking characters. Names beginning with the string “hla” or any string that would match (('H'|'h') ('L'|'l') ('A'|'a')), are reserved and also the string “na” or any string that would match (('N'|'n') ('A'|'a')), is reserved. These can not be included as a user-defined name. We check this constraint for object class names, interaction class names, attribute names, parameter names, datatype names, enumerated datatype enumerators, enumerated datatype values, fixed record field names, variant record alternative names, basic data representation names, dimension names, transportation type names, synchronization point names, note identifying labels. Other names could be checked in the same fashion if needed.

Another common constraint is the cardinality constraint. We check the cardinalities in two ways: the upper bounds and the lower bounds. Upper bounds cause error messages and the user is not allowed to do the operation. Lower bounds cause warnings, allowing the user to add the needed elements later.

Unique name constraints are also applicable for various elements. As the name suggests if an element’s name needs to be unique throughout the project, this constraint causes error messages for duplicate names. We also checked the null references, in order to catch the forgotten elements. The null reference constraint causes warnings.

In our metamodel some elements have default attributes or names, which can not be changed. For example, the name of the object class “HLAobjectRoot” cannot be changed. Such violations cause errors or warnings according to the severity of the

situation. Some attributes and some model parts are mandatory, and the user is expected to fill these fields. Some attributes may be "NA" which means not applicable. Some attributes have a common format like date, email …etc. We also check the validity of attributes and model parts.

All of the constraints are written in OCL [6]. Extended OCL support of the GME helped us immensely to enhance the precision of the metamodel. Succeeding sections describe the Federation Design Model, Object Model and OMT Core Folder in detail.

4.1.1 Federation Design Model

The Federation Design Model (FDM) provides an interface to define a federation and the federates and to connect them to the related FOM and SOMs. In FDM, FOM and SOMs are referenced object models. Each design can include one federation and one FOM reference, while there may be any number of federates and SOMs.

Figure 4.1 shows the GME class diagram of federation design model. There is a "MemberOf" connection between federation and federates. This connection presents the federation execution capabilities. Federation Design Model is not intended to complete; it is constructed to show the usage of object models in context.

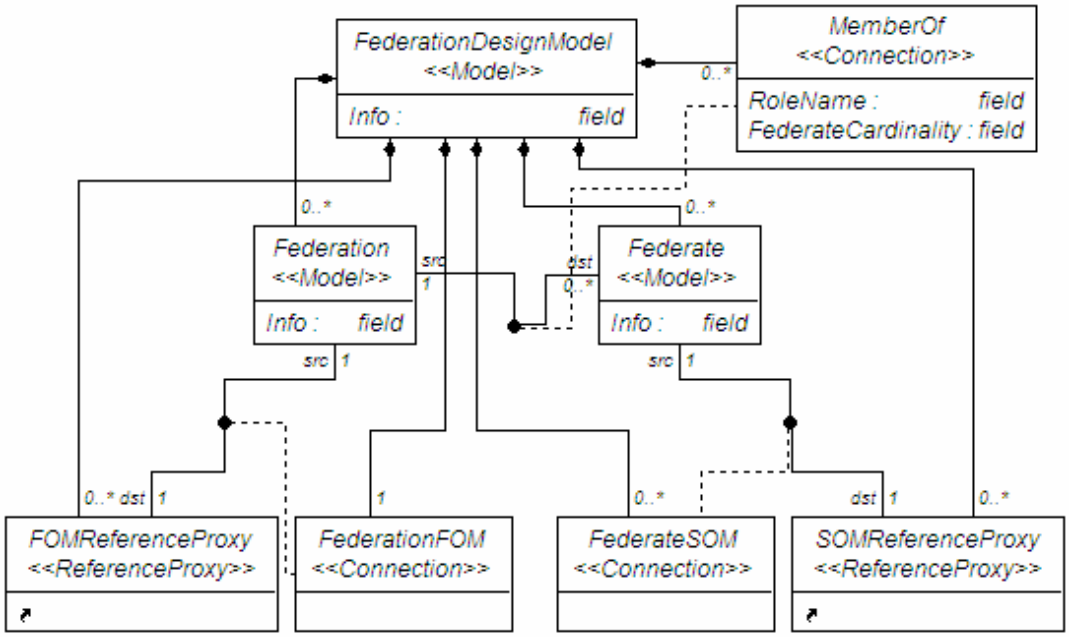

Figure 4.1: Federation design model.

GME provides import and export facilities with XML and XME (XML Extension) files. Figure 4.2 shows a sample XME output for Federation Design Model.

```
<model id="id-0065-000000c6" kind="FederationDesignModel">
   <name>Federation Design Model</name>
   <model id="id-0065-000000c7" kind="Federation">
      <name>Restaurant Federation</name>
   </model>
   <model id="id-0065-000000c8" kind="Federate">
      <name>Restaurant Federate 1</name>
   </model>
   <model id="id-0065-000000c9" kind="Federate">
      <name>Restaurant Federate 2</name>
   </model>

   <reference id="id-0067-00000059" kind="SOMReference"
referred="id-0065-00000001">
      <name>SOMReference</name>
   </reference>
   <reference id="id-0067-0000005a" kind="SOMReference"
referred="id-0065-00000001">
      <name>SOMReference</name>
   </reference>
   <reference id="id-0067-0000005b" kind="FOMReference"
referred="id-0065-000000ca">
      <name>FOMReference</name>
   </reference>

   <connection id="id-0068-0000005b" kind="FederateSOM">
      <connpoint role="src" target="id-0065-000000c8"/>
      <connpoint role="dst" target="id-0067-00000059"/>
   </connection>
   <connection id="id-0068-0000005c" kind="FederateSOM">
      <connpoint role="src" target="id-0065-000000c9"/>
      <connpoint role="dst" target="id-0067-0000005a"/>
   </connection>
   <connection id="id-0068-0000005d" kind="FederationFOM">
      <connpoint role="src" target="id-0065-000000c7"/>
      <connpoint role="dst" target="id-0067-0000005b"/>
   </connection>

   <connection id="id-0068-0000005e" kind="MemberOf">
      <name>MemberOf</name>
      <attribute kind="FederateCardinality" status="meta">
         <value>0..*</value>
      </attribute>
      <attribute kind="RoleName" status="meta">
         <value>member</value>
      </attribute>
      <connpoint role="src" target="id-0065-000000c7"/>
      <connpoint role="dst" target="id-0065-000000c8"/>
   </connection>
```

```
   <connection id="id-0068-0000005f" kind="MemberOf">
      <name>MemberOf</name>
         <attribute kind="FederateCardinality">
            <value>0..*</value>
         </attribute>
         <attribute kind="RoleName" status="meta">
            <value>member</value>
         </attribute>
         <connpoint role="src" target="id-0065-000000c7"/>
         <connpoint role="dst" target="id-0065-000000c9"/>
      </connection>
   </model>
</folder>
```

Figure 4.2: XME output for Federation Design model.

4.1.2 Object Model

The Object Model paradigm sheet includes the main diagram for object models. As seen in Figure 4.3, there are three types of object models, namely, FOM, SOM and Other. FOM and SOM are HLA object models which are defined in HLA OMT specification. The "Other" type provides a template for "temporary" object models -not to be included in Federation Design model.

Object Model, which is the parent of FOM, SOM and Other, is an abstract class. The inheritance operator in this figure presents a parent-child relation that is analogous to the inheritance in usual OO approach. Object models have some attributes, most of which correlate with the object model identification table categories in [11]:

- **Name**: The name assigned to the object model.
- **Version**: The version identification assigned to the object model.
- **Modification Date**: The latest date on which this version of the object model was created or modified.
- **Purpose**: The purpose for which the federate or federation was developed.
- **Application Domain**: The type or class of application to which the federate or federation applies.
- **Sponsor**: The organization that sponsored the development of the federate or federation.

- **Point of Contact**: The name of the point of contact (POC) for information.
- **POC Organization**: The organization with which the POC is affiliated.
- **POC Telephone**: The telephone number for the POC.
- **POC E-mail**: The e-mail address of the POC.
- **References**: Additional sources of information. The default value is “NA”.
- **Other**: Other information related to the object model. The default value is “NA”.
- **MOM version**: The version of the included management object model. The default value is “IEEE 1516”. Selecting a MOM name is required for all FOMs. In SOMs the MOM shall be included if needed.
- **Notes**: Note labels added to the object model.

Object models have five aspects, namely Classes, User-Supplied Tags, Synchronization, Switches and Time Representation. In each aspect the model definitions are taken from the related paradigm sheets with proxy elements. The Classes aspect includes the definition of object classes, attributes, interaction classes and parameters. The User-Supplied Tags aspect includes the user-supplied tag elements. The Synchronization aspect includes the definition of synchronization point models. The Switches aspect includes the switches in order to change the initial settings and time representation aspect includes the definition of lookahead and timestamp models.

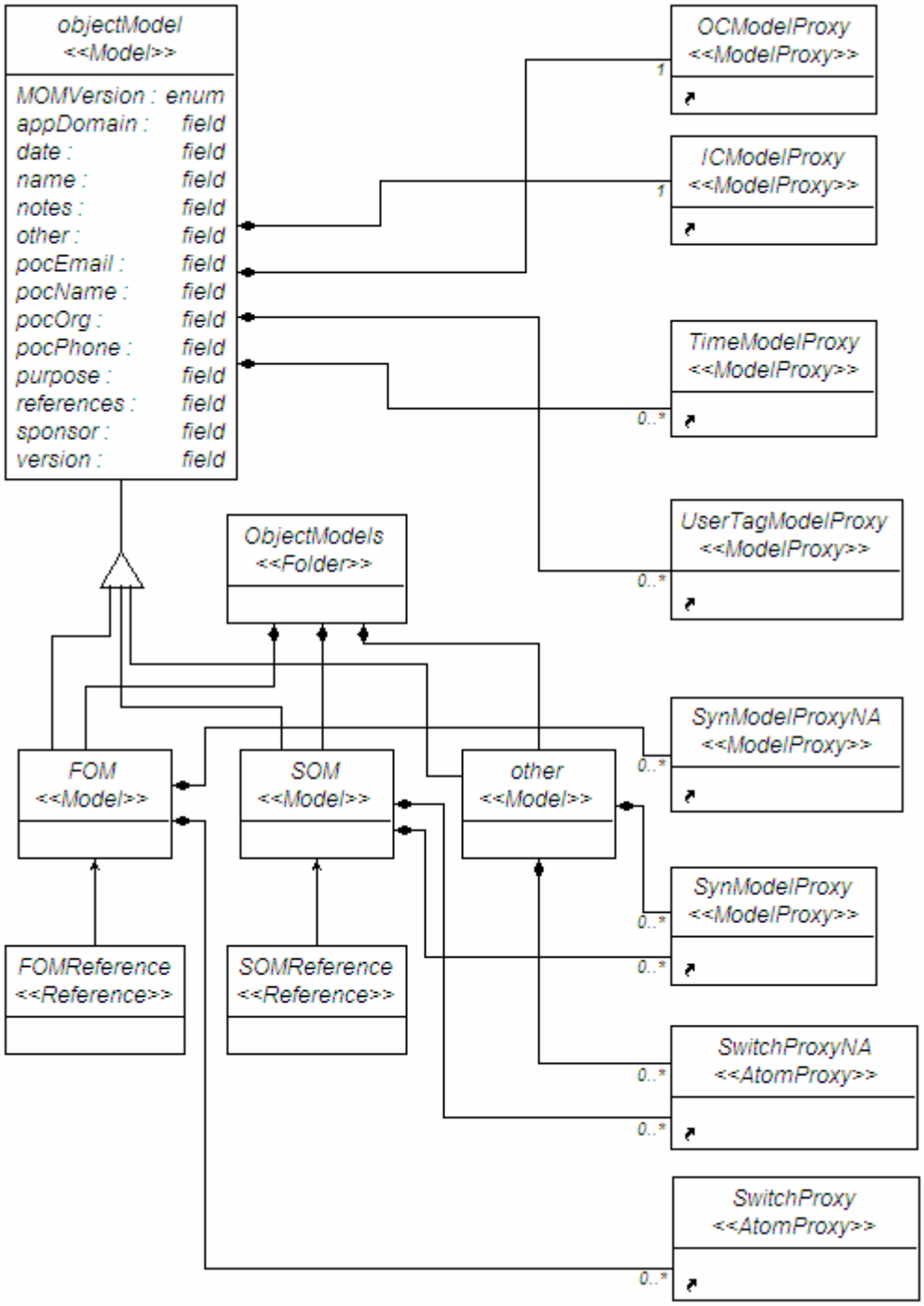

Figure 4.3: Object Models diagram.

A sample XME file that includes the object model information is shown in Figure 4.4.

```
<folder id="id-006a-00000002" kind="ObjectModels">
   <name>Object Models</name>
   <model id="id-0065-00000001" kind="SOM">
      <name>RestaurantSOM</name>
      <attribute kind="MOMVersion">
         <value>ieee</value>
      </attribute>
      <attribute kind="appDomain">
         <value>Restaurant operations</value>
      </attribute>
      <attribute kind="date">
         <value>1998-01-01</value>
      </attribute>
      <attribute kind="name">
         <value>Restaurant Example</value>
      </attribute>
      <attribute kind="notes" status="meta">
         <value></value>
      </attribute>
      <attribute kind="other">
         <value>See Mobil Int. Restaurant Guide</value>
      </attribute>
      <attribute kind="pocEmail">
         <value>doej@fedfoods.com</value>
      </attribute>
      <attribute kind="pocName">
         <value>Mr. Joseph Doe</value>
      </attribute>
      <attribute kind="pocOrg">
         <value>Joe&amp;apos;s Place</value>
      </attribute>
      <attribute kind="pocPhone">
         <value>1-977-555-1234</value>
      </attribute>
      <attribute kind="purpose">
         <value>Object model for a restaurant fed.</value>
      </attribute>
      <attribute kind="references">
         <value>www.fedfoods.com/restsim.html</value>
      </attribute>
      <attribute kind="sponsor">
         <value>Federated foods</value>
      </attribute>
      <attribute kind="version">
         <value>1.0 Alpha</value>
      </attribute>
      <model id="id-0065-00000002" kind="objects">
         <name>objects</name>
      </model>
      <model id="id-0065-00000003" kind="interactions">
         <name>interactions</name>
      </model>
   </model>
</folder>
```

Figure 4.4: XME output of object model information.

4.1.3 OMT Core Elements

The OMT Core folder includes basic elements needed to define an object model; it serves classification purposes. OMT Core folder includes classes, data types, dimensions, normalization functions, notes, switches, synchronization points, user supplied tags, time representations and transportations models. The following sections describe the models in this folder.

4.1.3.1 Classes

The Classes paradigm sheet provides object class, interaction class, attribute, and parameter definitions for the object models. It refers to object class structure table, interaction class structure table, attribute table, and parameter table in HLA OMT specification.

Figure 4.5 shows the elements defined in this paradigm sheet. “OMTClass”, which is the parent of “objectClass” and “interactionClass”, is an abstract class. “HLAobjectRoot” and “HLAinteractionRoot” are the default classes. Each one is defined in the model with an appropriate inheritance type.

GME features three types of inheritance: normal inheritance, implementation inheritance and interface inheritance. In implementation inheritance (denoted by a black dot inside the inheritance icon), the subclass inherits all of the base class' attributes, except for those containment associations where the base class functions as the container Interface inheritance (denoted by an unfilled circle) allows no attribute inheritance but does allow full association inheritance, with one exception: containment associations where the base class functions as the container are not inherited.

The same explanation applies for “OMTAttribute”, which is an abstract class for attributes and parameters. “HLAprivilegetoDeleteObject” is a default attribute for “HLAobjectRoot” object class and it is also defined with implementation inheritance.

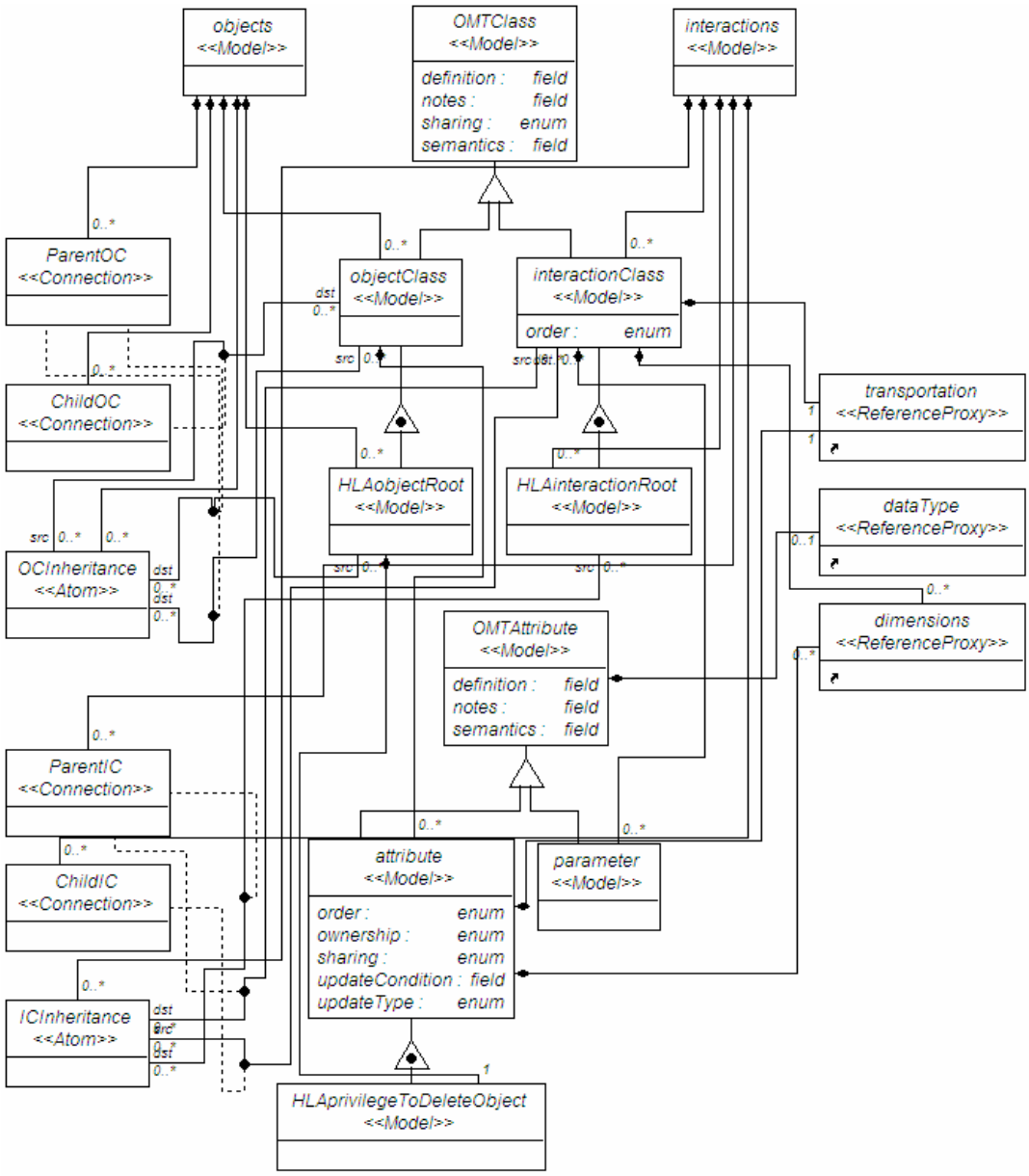

Figure 4.5: Classes diagram.

The HLA object and interaction classes support only single inheritance: Each class has at most one immediate superclass, and loops are not allowed. The HLA object class structure and the HLA interaction class structure are defined with class elements and inheritance operators. “OCInheritance” is used for object class hierarchy and “ICInheritance” is used for interaction class hierarchy. Subclasses can be considered to be specializations, or refinements, of their superclasses. “HLAobjectRoot” is the superclass of all other object classes and it may have attributes like other object classes. “HLAinteractionRoot” is the superclass of all other interaction classes and it may have

parameters like other interaction classes. Individual class names need not be unique. But the names of the sibling classes must be different. A violation of this constraint causes an error message.

Object classes have attributes and interaction classes have parameters. The names assigned to attributes of any particular object class shall not duplicate (overload) the names of attributes of this class or any higher level superclass. In the same way, the names assigned to parameters of any particular interaction class shall not duplicate (overload) the names of parameters of this class or any higher level superclass. Any violation of these constraints causes error messages. Be reminded that in case of an error message the user cannot complete the attempted operation. In contrast, a warning message allows the operation to complete, with the presumption that the user will have the violation removed from the final model.

The models in this paradigm sheet have some attributes and parts. We will look at the most important ones.

OMTClass Model Attributes:

- **Sharing**: Each object class or interaction class shall have information on publication and subscription capabilities. Valid entries for sharing shall be: “P (Publish)”, “S (Subscribe)”, “PS (PublishSubscribe)” and “N(Neither)”.
- **Definition**: Information about the class. The FOM/SOM lexicon defined in HLA OMT specification can be derived from this attribute.
- **Semantics**: Semantics for the class if needed.
- **Notes**: Note labels added to the class.

ObjectClass Model Attributes

No additional attributes.

InteractionClass Model Attributes

- **Order**: Specifies the order of delivery to be used. Valid values are: “Receive” and “TimeStamp”.

- **Dimension**: Available dimensions are provided with references that shall be referred to any dimension described in dimensions folder. A value of “NA” may also be provided with no reference.
- **Transportation**: Specifies the type of transportation to be used. Transportation is provided with a reference that shall be referred to any transportation described in transportations folder. Each interaction class has one and only one transportation type. Default transportations “HLAbestEffort” and “HLAreliable” are available in IEEE default library which is prepared with our proposed metamodel.

OMTAttribute Model Attributes:

- **Datatype**: Identify the datatype of the attribute. Datatype is provided with a reference that shall be referred to any simple, enumerated, array, fixed record, or variant record datatype described under datatypes folder. A value of “NA” may also be provided with no reference. When datatype is “NA”, the update type, update condition, and available dimensions shall also be “NA”.
- **Definition**: Information about the attribute or parameter. The FOM/SOM lexicon defined in HLA OMT specification can be derived from this attribute.
- **Semantics**: Semantics for the attribute or parameter if needed.
- **Notes**: Note labels added to the attribute or parameter.

Attribute Model Attributes

- **Dimension**: Available dimensions are provided with references in the same way with interaction classes.
- **Transportation**: Specifies the type of transportation to be used; provided in the same way with interaction classes.
- **UpdateType**: The policy for updating an instance of the class attribute. Valid values are: “Static”, “Periodic”, “Conditional” and “NA”.
- **UpdateCondition**: Expand and explain the policies for updating an instance of the class attribute. If the update type is “Static” or “NA”, “NA” shall be entered.
- **DivestAcquire**: Indicates whether ownership of an instance of the class attribute can be divested or acquired. Valid values are: “D (Divest)”, “A (Acquire)”, “N

(NoTransfer)" and "DA (DivestAcquire)". In a FOM, if an instance attribute can be divested by a federate, it should be acquirable by some other federate in the federation. Therefore, the unique designations for this column shall be: "N (NoTransfer)" or "DA (DivestAcquire)".

- **Sharing**: Each attribute shall have information on publication and subscription capabilities, provided in the same way with OMT classes.

Parameter Model Attributes

No additional attributes.

A part of a sample XME file that includes an object class definition is shown in Figure 4.6.

```
<model id="id-0065-00000008" kind="objectClass">
   <name>Employee</name>
   <attribute kind="definition">
      <value>A person working for the restaurant</value>
   </attribute>
   <attribute kind="notes" status="meta">
      <value></value>
   </attribute>
   <attribute kind="semantics" status="meta">
      <value></value>
   </attribute>
   <attribute kind="sharing" status="meta">
      <value>N (Neither)</value>
   </attribute>
   <model id="id-0065-00000053" kind="attribute">
      <name>PayRate</name>
      <attribute kind="definition" status="meta">
         <value></value>
      </attribute>
      <attribute kind="notes">
         <value>Note1</value>
      </attribute>
      <attribute kind="order" status="meta">
         <value>Time Stamp</value>
      </attribute>
      <attribute kind="ownership">
         <value>divestacquire</value>
      </attribute>
      <attribute kind="semantics" status="meta">
         <value></value>
      </attribute>
      <attribute kind="sharing">
         <value>PS (PublishSubscribe)</value>
      </attribute>
```

```
      <attribute kind="updateCondition">
         <value>Merit increase * [1,2]</value>
      </attribute>
      <attribute kind="updateType">
         <value>conditional</value>
      </attribute>
      <reference                    id="id-0067-00000009"
kind="transportationRef" referred="id-0066-0000003b">
      </reference>
      <reference   id="id-0067-0000000a"   kind="datatype"
referred="id-0065-0000005c">
      </reference>
   </model>
   <model id="id-0065-00000079" kind="attribute">
      <name>HomeNumber</name>
      .
      .
   </model>
   <model id="id-0065-0000007a" kind="attribute">
      <name>HomeAddress</name>
      .
      .
   </model>
   <model id="id-0065-0000007b" kind="attribute">
      <name>YearsOfService</name>
      .
      .
   </model>
</model>
```

Figure 4.6: The XME output for an object class.

4.1.3.2 Dimensions

Dimensions model map to dimension table of HLA OMT tabular format. Figure 4.7 shows the dimension diagram. Federations use dimensions to limit the delivery of some data on the basis of object class, interaction class, and object attribute. They refer to the available dimensions using “dimensionRef” model element. Each set of available dimensions is a subset of all dimensions, so there can be many referred dimensions. Dimensions folder is in the root folder of the project, so that dimensions are shared models.

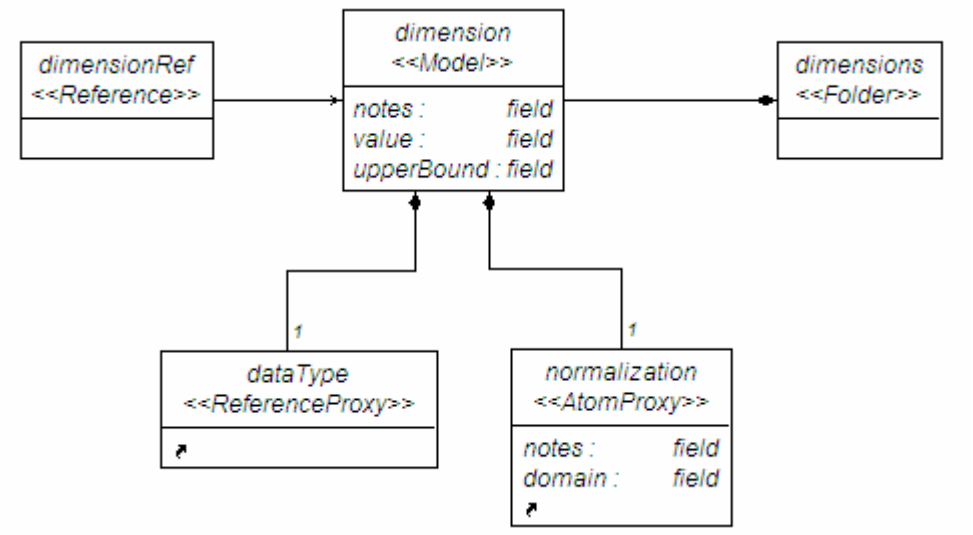

Figure 4.7: Dimension diagram.

The important attributes of dimension model are "value" and "upperBound". Dimension upper bound specifies the upper bound for the dimension as a positive integer. "Value" specifies the default range for the dimension.

Each dimension should have a datatype reference, this can refer to a simple datatype or an enumerated datatype. Each dimension has a normalization function that specifies the map from a dimension's bounding coordinates to nonnegative integer sub-ranges in the range [0, dimension upper bound). The following normalization functions are commonly used and they are provided with our metamodel; new functions can be referred as well:

- linear(domain, dimensionLower, dimensionUpper)
- linearEnumerated(domain, mappedSet)
- enumeratedSet(domain, mappedSet)
- logarithmic(domain, domainLower, domainUpper)
- tanh(domain, domainCenter, domainSize)
- newFunction(parameter1, parameter2, parameter3)

Figure 4.8 denotes the normalization function diagram, where the above functions are modeled.

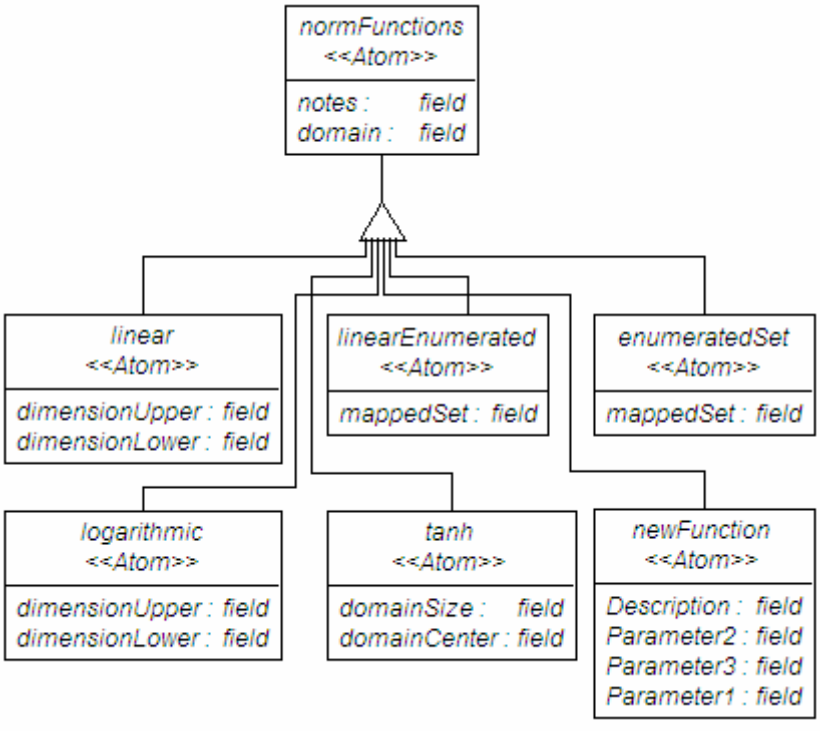

Figure 4.8: Normalization functions.

Unfortunately the format of the default range for dimensions and the format of some parameters for normalization functions are not checked as these fields accept arbitrary strings.

4.1.3.3 Time Representation

Federates may associate themselves or some of their activities with points on the HLA time axis, i.e. time stamps. It is also important to define the lookahead characteristics of federates and federations in order to provide compatibility. Time stamp and lookahead are modeled in time representation diagram shown in Figure 4.9. Both of them may have a datatype reference, which can refer to a simple, enumerated, array, fixed record or variant record datatype.

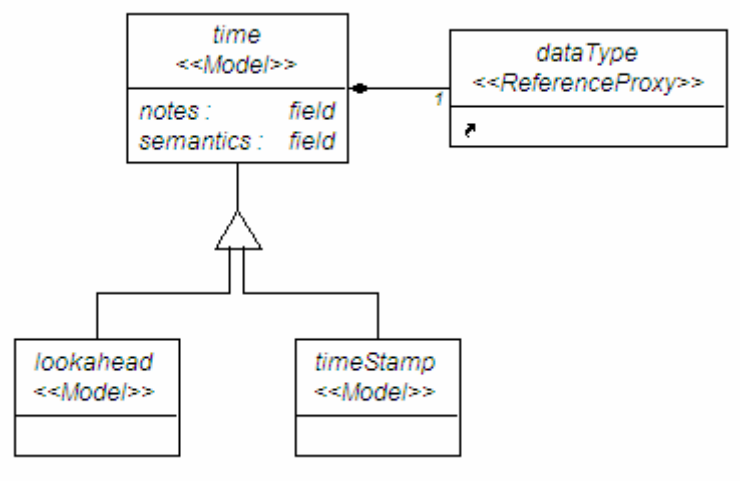

Figure 4.9: Time representation model.

4.1.3.4 User Supplied Tags

With the User Supplied Tags diagram, the mechanism for federates to supply tags with some certain HLA services is modeled. The HLA service categories that are capable of accepting a user-supplied tag are: update/reflect instance attribute values, send/receive an interaction, delete/remove an object instance, divestiture request, divestiture completion, acquisition request, and request update services.

Figure 4.10 shows the user supplied tags model. Each service category may have a datatype reference which can refer to a simple, enumerated, array, fixed record or variant record datatype.

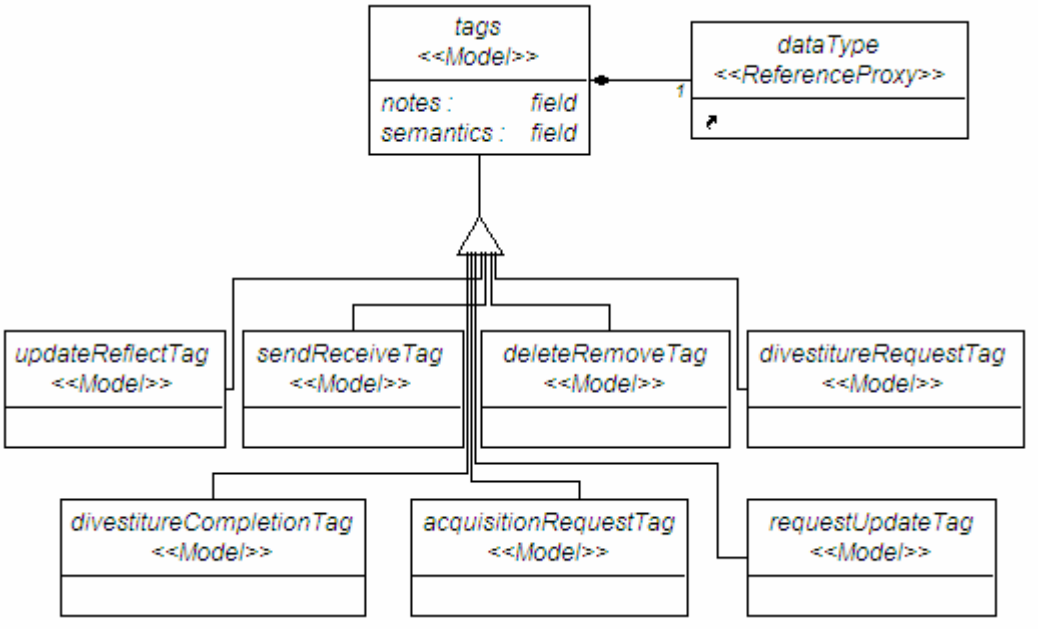

Figure 4.10: User-supplied tags diagram.

4.1.3.5 Synchronizations

The synchronizations diagram shown in Figure 4.11 defines synchronization points to synchronize federation activities. Each synchronization point has a tag datatype reference that identifies the datatype of the user-supplied tag when needed. It can refer to a simple, enumerated, array, fixed record or variant record datatype.

For SOMs, synchronization points have “notes” attribute, “semantics” attribute, and an attribute named capability that indicates the level of interaction that a federate is capable of honoring. Valid values for capability are: “Register”, “Achieve”, “RegisterAchieve” and “NoSynch”. For FOMs, synchronization points have only “notes” and “semantics” attributes. In the diagram “synchronization” model is used for SOMs and “synchronizationNA” model is used for FOMs.

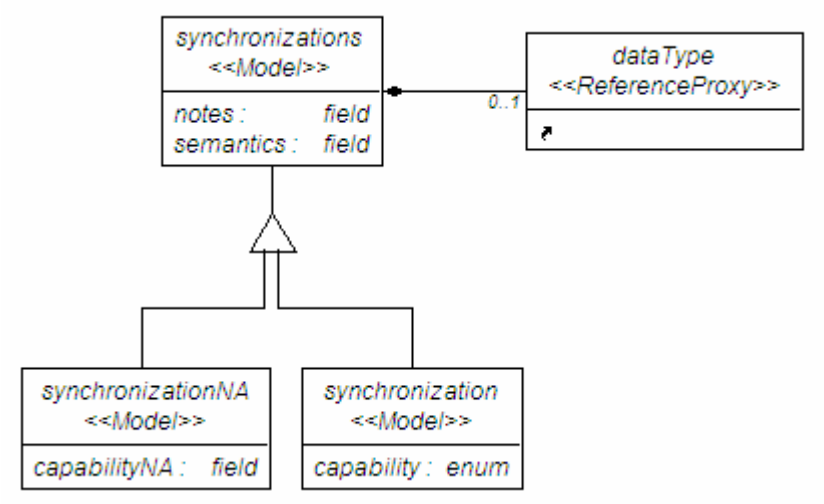

Figure 4.11: Synchronizations diagram.

4.1.3.6 Transportations

This diagram defines transportation types that define the transportation of data among federates. Two transportation types, “HLAreliable” and “HLAbestEffort”, are required by the HLA and these are provided with IEEE default library model. This library is described in section 4.2.1.

Figure 4.12 shows the transportations diagram. Transportation folder is in the root folder of the project, so transportations are shared models. Object class attributes and interaction classes refer to a transportation type via the “transportationRef” model element.

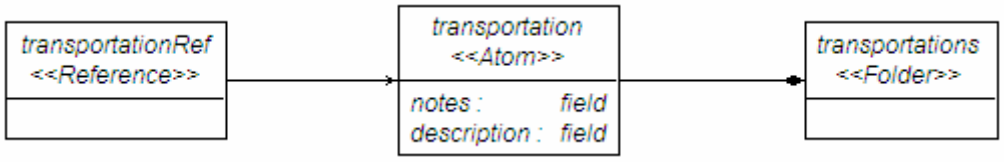

Figure 4.12: Transportations diagram.

4.1.3.7 Switches

The Switches diagram provides initial settings of some actions provided on behalf of federates. Researchers who will do future work should note that although the initial setting of each switch is specified, the value of each switch may be changed during execution. The switches whose setting can be provided are: auto provide, convey region designator sets, attribute scope advisory, attribute relevance advisory, object class relevance advisory, interaction relevance advisory, and service reporting. For SOMs, setting can be "NA" as well as enabled or disabled, but for FOMs, all switches must be (initially) set as enabled or disabled.

4.1.3.8 Data types

Figure 4.13 denotes the datatypes diagram. "DatatypeModel" is an abstract model on the top of other datatypes. There is a "basicData" model to provide the representation type of some datatypes. Basic data representation is not used as a datatype, but it forms the basis of the datatypes. The "simpleData" model can be used to describe simple, scalar data items. The "enumeratedData" model can be used to describe data elements that can take on a finite discrete set of possible values. The "arrayData" model can be used to describe indexed homogenous collections of datatypes also known as arrays or sequences.

The "fixedRecordData" model can be used to describe heterogeneous collections of types also known as records or structures. Each fixed record datatype may contain fields that are of other types, such as simple datatypes, fixed records, arrays, enumerations, or variant records.

The "variantRecordData" model can be used to describe discriminated unions of types known as variant or choice records. Each variant record datatype may contain enumerators that determine the alternatives.

Datatypes folder is in root folder in the project. Datatypes are shared models and several models (object class attributes, interaction class parameters, dimensions, time representations, user-supplied tags, and synchronization points) refer to datatypes.

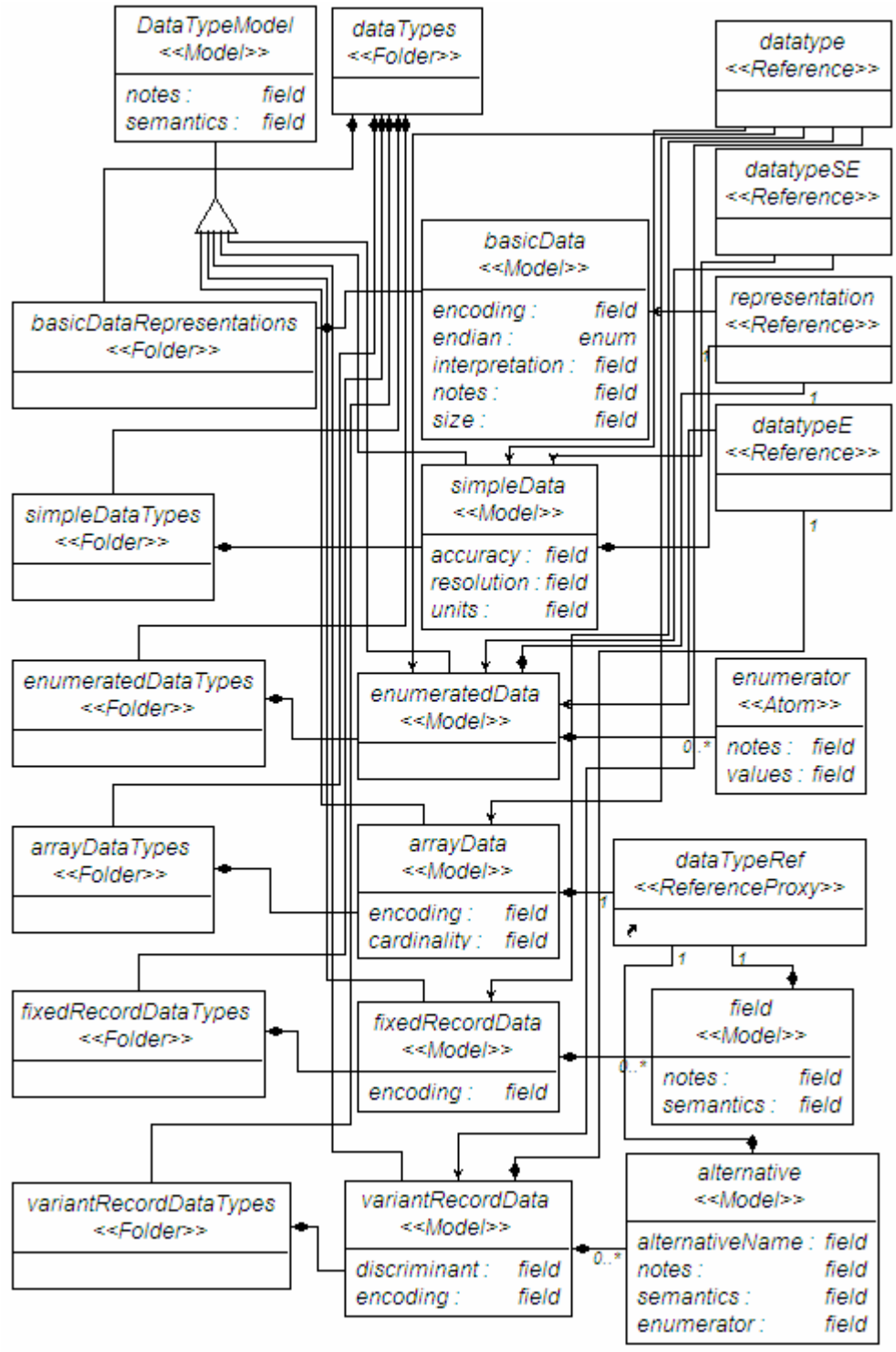

Figure 4.13: Datatypes diagram.

A predefined set of basic data representations, three predefined simple datatypes, a single predefined enumerated datatype and three predefined array datatypes are defined in IEEE default library mode. This library is explained in section 4.2.1.

4.1.3.9 Notes

The Notes diagram provides note elements for the object model elements. Notes are defined under the root folder and they can be pointed out from any model element described previous sections. A note may be referenced any number of times and a single object model element may have any number of notes.

Also additional notes, related to the design model, can be added by the user with GME annotation facilities. These annotations are not a part of the object model.

4.2 Object Model Design Environment

After defining the metamodel as a GME paradigm, a design environment for object models can be automatically generated by using the GME Meta 2004 Interpreter. The Object Model Design Environment (OMDE) is explained in Appendix A as the User's Guide. This chapter explains IEEE default library, the MOM and an example federation design model.

4.2.1 IEEE Defaults Library

An important facility that GME needs is default model elements that mean when the user creates a model some model elements are already defined for it. The GME overcomes this problem with libraries. The user defines libraries and attaches them to the model where needed. So in order to define default datatypes and transportations for HLA, we modeled IEEE defaults library with our metamodel. To utilize these datatypes and transportations, first he should attach this library to his model.

The predefined basic data representations and datatypes in this library are listed in Figure 4.14 in the tree browser of GME for IEEE defaults library. The XME output, generated by GME, of this library is reproduced in Appendix B.

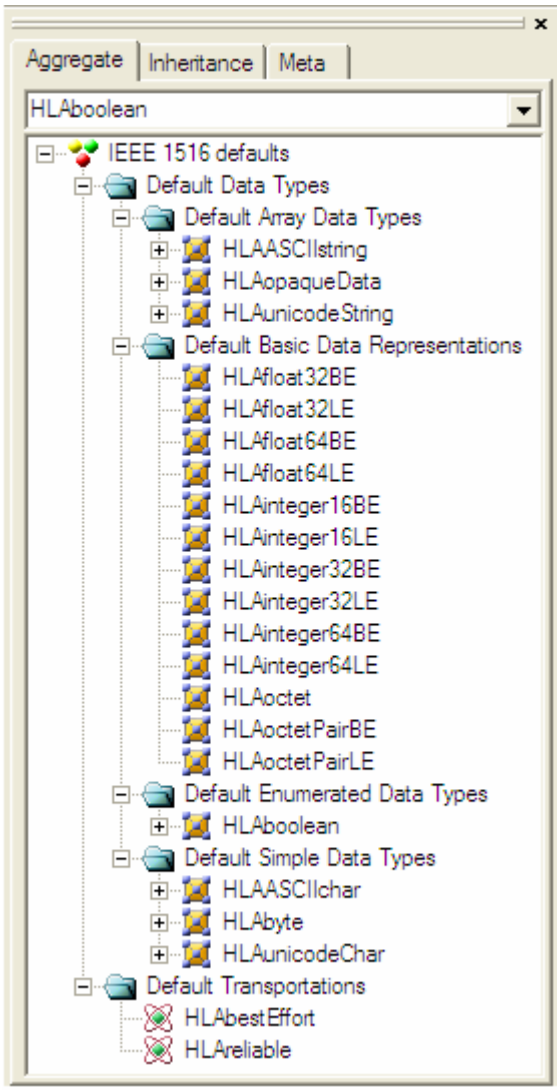

Figure 4.14: Default datatypes and transportations.

4.2.2 Management Object Model

Management Object Model, s defined in Federate Interface Specification. MOM provides facilities for joined federates to access RTI services during federation execution. MOM specification employs the OMT tabular format and so we can readily model it with our metamodel. MOM provides some default model elements for some services, such as publishing object classes; registering object instances and updating values of attributes of those object instances; subscribing to and receiving some interactions; or publishing and sending other interactions. The classes, attributes,

parameters, datatypes and dimensions in MOM are modeled completely. Figure 4.15 shows the datatypes and dimensions in the tree browser.

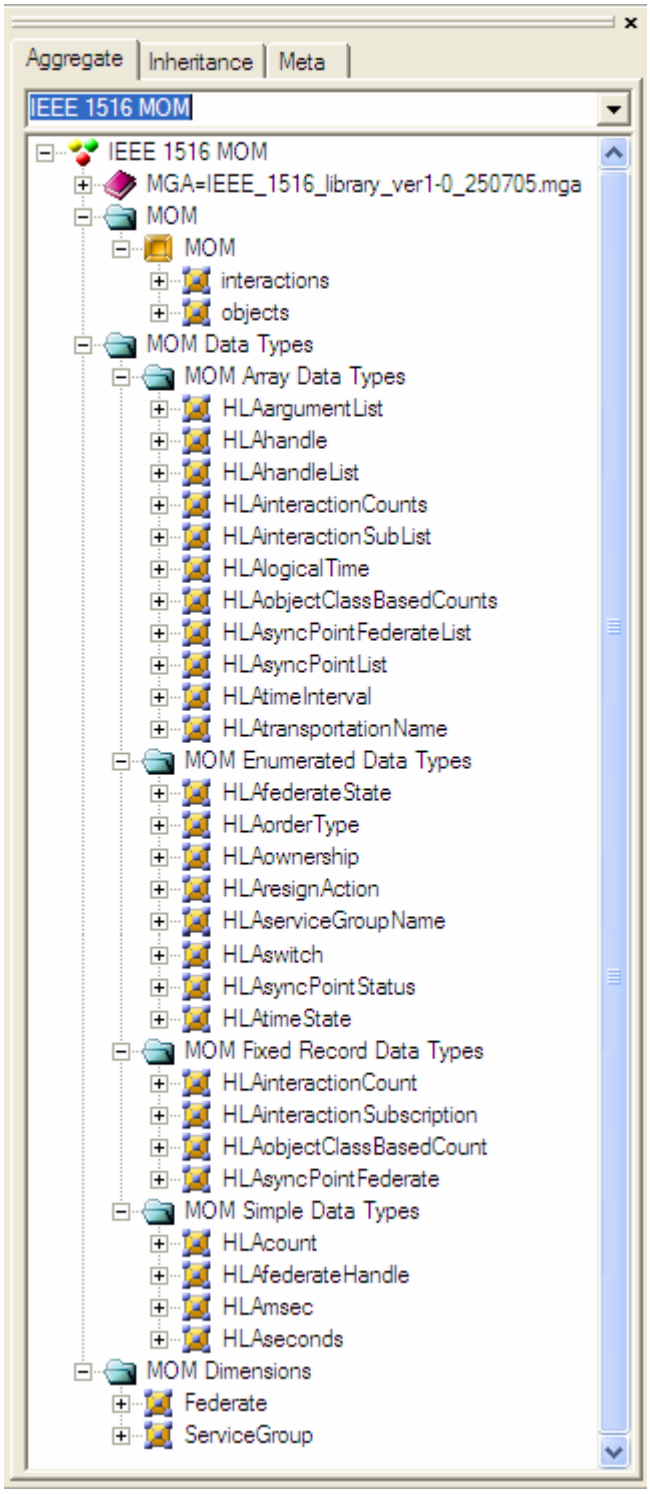

Figure 4.15: MOM datatypes and dimensions.

Although GME provides XME outputs, which has an extended XML representation, the size of the files can get very large. MOM has more than 10,000 XME lines, where our proposed metamodel has over 40,000. GME also needs improvement on its import and export mechanisms. Because of this reason we cannot show the XME output for MOM.

Figure 4.16 shows the object classes and Figure 4.17 shows the interaction classes drawn with the HLA OM design environment. We refer to HLA federate interface specification for further details.

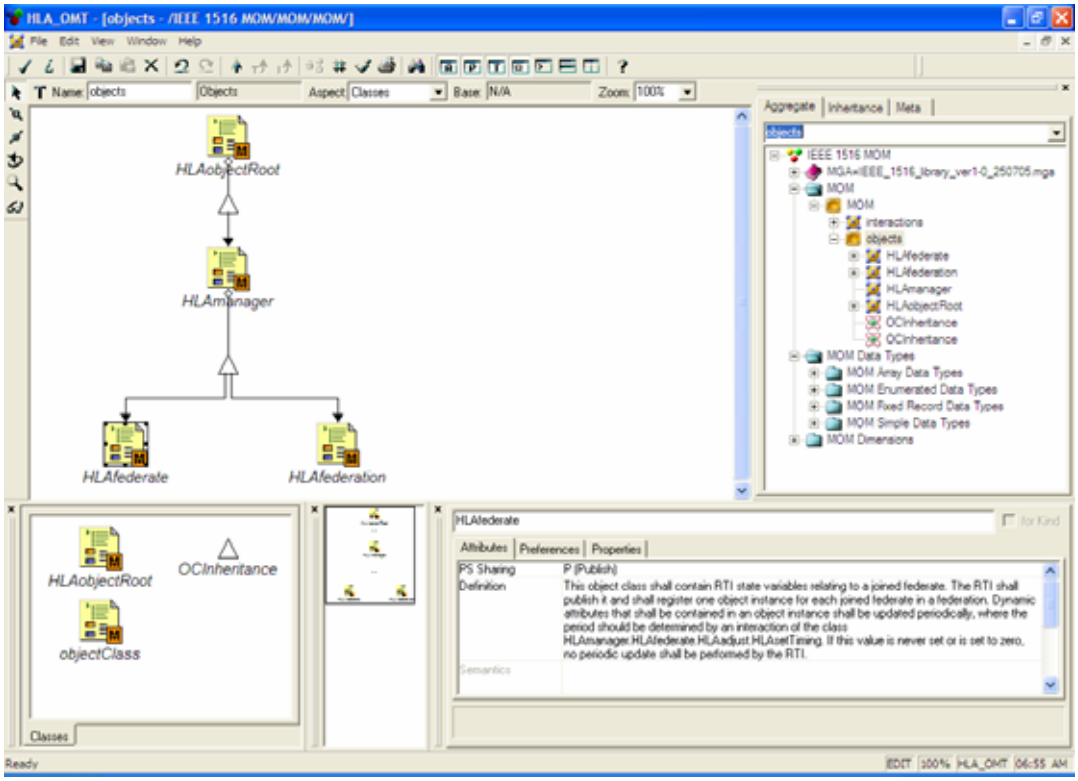

Figure 4.16: MOM object classes.

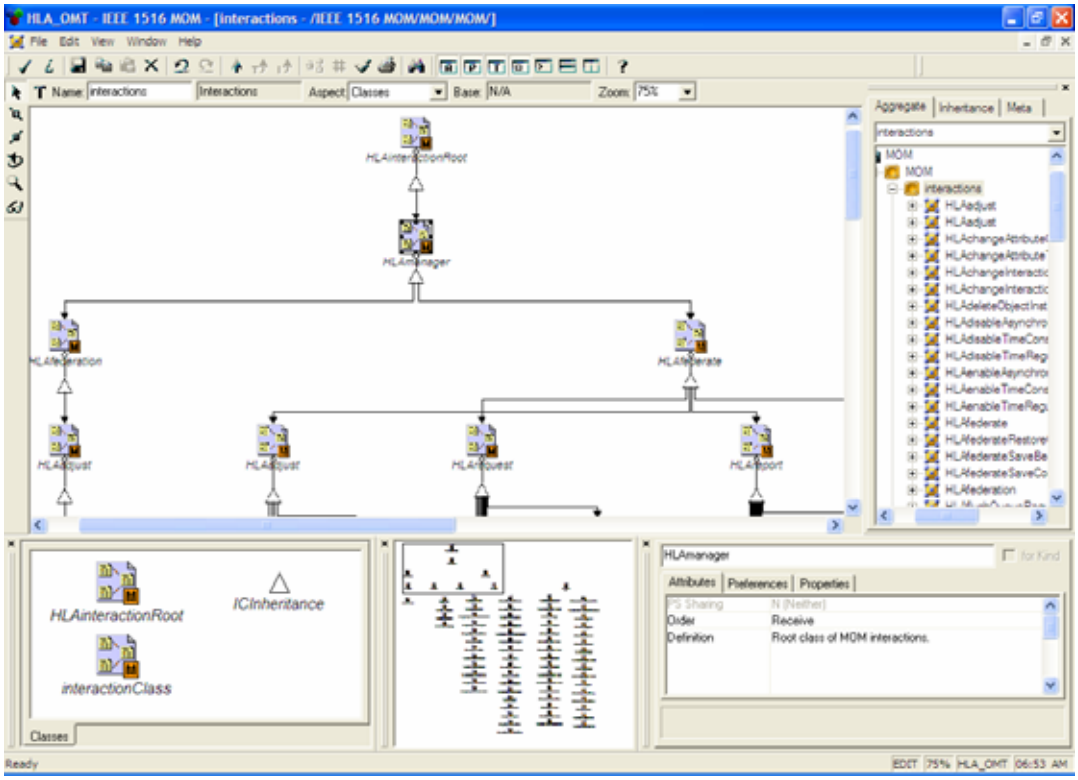

Figure 4.17: MOM interaction classes.

4.2.3 Sample Federation Design Model

We have modeled a sample restaurant federation design model, which is a well known example, since the HLA standard gives this example in order to explain object model template tabular format. Here we extend the given example with a few new modeling elements and a simple federation design model. All sample tables included in the standard document are completely covered in this example.

Figure 4.18 shows a screenshot from the modeling environment. On the right side of the figure the tree browser shows the datatypes, dimensions and transportations. On the left side of the figure, above model shows the federation design model with one federation and two federates connected to the related FOM and SOMs; the following model shows some of the interaction classes of Restaurant Federate SOM.

Figure 4.19 shows a sample object class hierarchy in Restaurant Federate SOM. This is only a part of the object class hierarchy with “HLAobjectRoot” on the top.

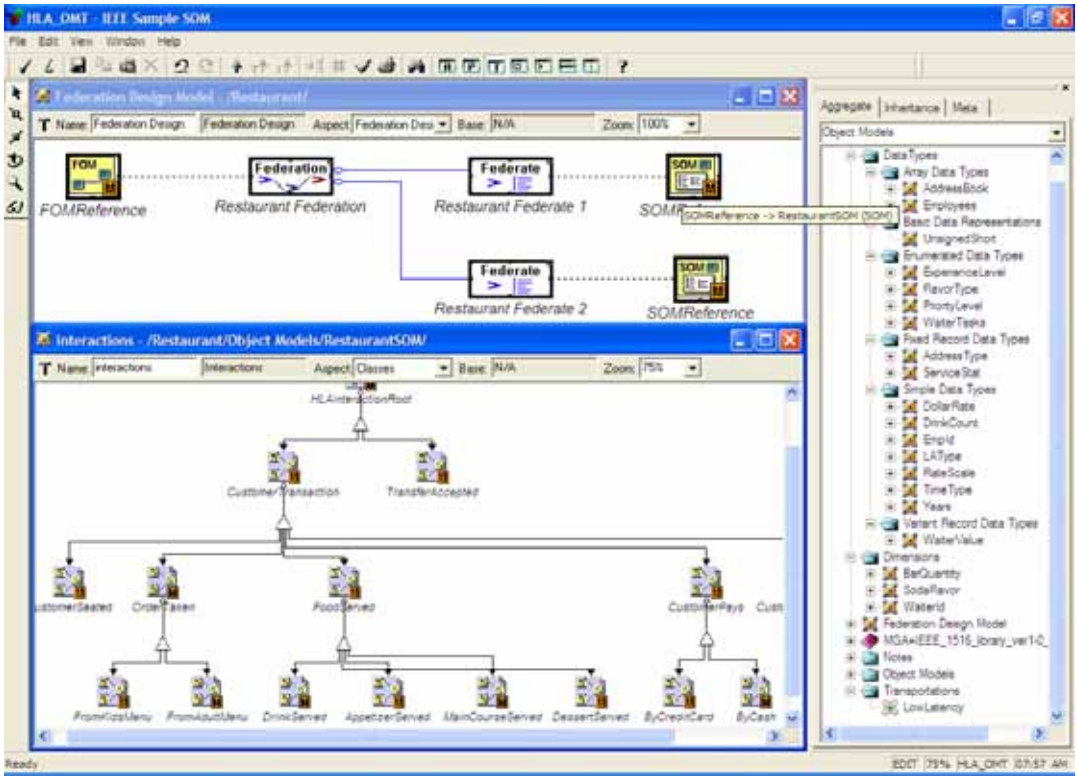

Figure 4.18: Sample federation design model overview.

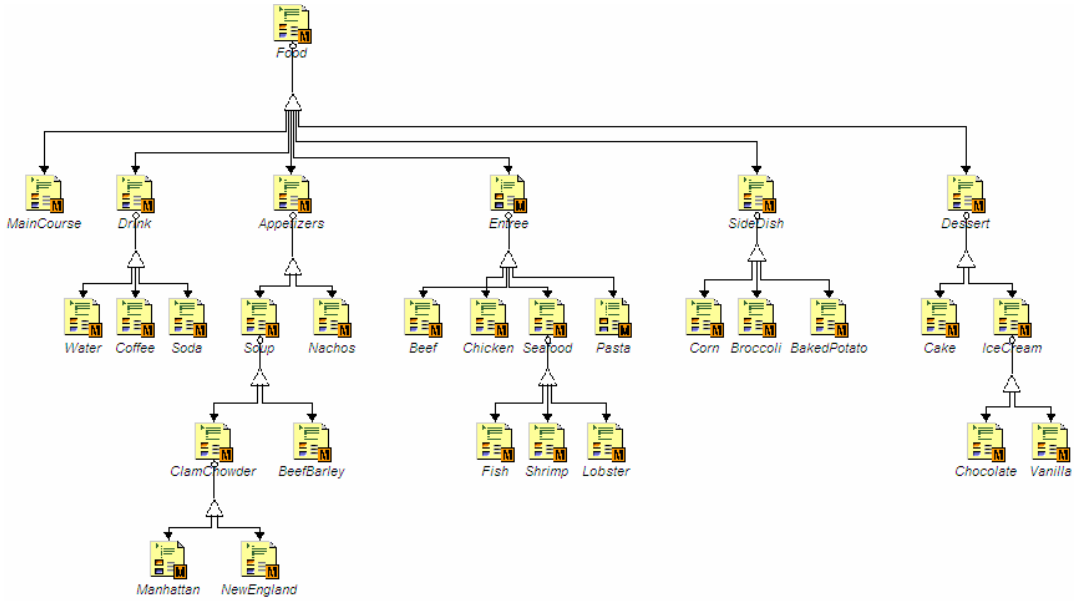

Figure 4.19: Sample object class hierarchy.

This example, along with the MOM model, raised our confidence that object model design environment created by GME based on our metamodel, is a sufficiently powerful visual tool for object models and it can be used for designing “real life” object models.

CHAPTER 5

CONCLUSION

This thesis introduces a metamodel as a GME paradigm for HLA object models and explains a design environment that is generated automatically. Our objective is to lay the foundation for developing a federation object model design tool, which can be a part of a federation development toolkit, by applying model integrated computing to HLA. Defining a metamodel for HLA OM can be seen as an initial step in this direction. This metamodel is used in GME's Meta interpreter and the Object Model Design Environment (OMDE) is generated automatically. The OMDE is a design tool to describe individual federates and federations with SOMs and FOMs respectively. OMDE also provides modeling simple federation designs with illustrative purposes. The object models defined with OMDE, are the domain models of the MIC process. So they can be used in another model transformation tool, such as code generators.

The proposed metamodel is fully compliant with IEEE Std. 1516.2, and it is strengthened with constraints written in OCL. OMT DIF formatted object models can be easily generated by writing simple interpreters. The OMT tabular format can also be generated easily, since all information in tabular format can map to a modeling element in our metamodel. Furthermore, our metamodel can support virtually any presentation of an object model compliant with IEEE Std. 1516. All that is required is to provide a generator based on model traversal, which is supported through an API in GME.

Although we have described a metamodel for HLA OM, services defined in federate interface specification can also be modeled in this way; and this work will make our metamodel more useful. Now, there is a metamodel that is ready to be used as a design tool and an input of a further study in MIC process.

5.1 Future Work

In this thesis we have used the latest version 4.11.10 of GME. Unfortunately, at this time it uses UML 1.4, it isn't upgraded to UML 2.0 yet. But we understand that the next generation modeling environment will support MOF 2.0 and also it will give opportunity to directly use MOF 2.0 for paradigms. So porting the HLA OM metamodel to MOF 2.0 should be considered.

Also the IEEE has announced that it began to revise IEEE 1516, IEEE 1516.1 and IEEE 1516.2 modeling and simulation standards on 31.03.2005. Upgrading this metamodel will be a future-work after the revision of the standards will be completed.

GME has a decorator facility for better visualization of the models. We used the standard Meta Decorator, but more advanced decorators can be defined and used. Providing more powerful visual elements will improve the object model design environment.

As we noted in related section, most of the attribute values are checked for syntactic validity while a few of them are not. Validation and also verification of the domain models will allow the metamodel to be used more effectively in development tools.

Applying MIC to HLA will bring new future works as model transformations and code generation, for which the proposed metamodel can be directly used. The researchers at Vanderbilt University developed a model-to-model transformation language The Graph Rewriting And Transformation language (GReAT) [46], and a meta-programmable transformation tool that supports the development of graphical language semantic translators using graph transformations. These translators can convert models of one domain into models of another domain. The GReAT tool can be used for model transformations.

Lastly this metamodel handles only HLA object models. A complete design and development environment for HLA based distributed simulations should be developed, supporting all steps of FEDEP. Subsequent tasks include a) generating HLA configuration files, b) importing and exporting object models represented in OMT tabular format, in OMT DIF format, or in some UML tool, c) defining a mapping from FOM elements to SOM elements, and d) producing Publish/Subscribe diagrams.

REFERENCES

[1] H.-E. Eriksson, M. Penker, B. Lyons, D. Fado, *UML 2 Toolkit,* Wiley, Indiana, 2004.

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

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

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

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

[6] J. Warmer, A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA, 2nd Ed.,* Addison-Wesley, Boston, 2004.

[7] B.P. Zeigler, H. Praehofer, T.G. Kim, *Theory of Modeling and Simulation, 2nd Ed.,* Academic Press, San Diego, 2000.

[8] *GME 4 User's Manual,* Institute for Software Integrated Systems at Vanderbilt University, 2004.

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

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

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

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

[13] *MDA Guide v1.0.1,* OMG, 2003.

[14] *MOF Specification v1.4,* OMG, 2002.

[15] *MOF 2.0 Core Specification,* OMG, 2003.

[16] *UML Specification v1.4,* OMG, 2001.

[17] *UML 2.0 Infrastructure Specification,* OMG, 2003.

[18] *UML 2.0 OCL Specification,* OMG, 2003.

[19] *Visual OMT 1516 User's Guide,* Pitch Technologies, Sweden, 2003.

[20] *AEgis Technologies,* http://www.aegistg.com, last accessed 26.07.2005.

[21] *DMSO web site,* https://www.dmso.mil, last accessed 30.07.2005.

[22] *IEEE web site,* http://www.ieee.org, last accessed 20.07.2005.

[23] *OMG web site,* http://www.omg.org, last accessed 26.07.2005.

[24] *Pitch Technologies,* http://www.pitch.se, last accessed 01.08.2005.

[25] *SIMplicity,* http://simplicity.calytrix.com, last accessed 01.08.2005.

[26] *XML,* http://www.w3.org/XML, last accessed 12.08.2005.

[27] J. Borah, *"Conceptual Modeling: The Missing Link of Simulation Development," in 2002 Spring Simulation Interoperability Workshop,* Orlando, 2002.

[28] J. Bourrely, P. Carle, M. Barat, F. Lévy, *"GENESIS: an integrated platform for designing and developing HLA applications," in EURO SIW 2005,* France, 2005.

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

[30] P. Desfray, *"UML Profiles versus Metamodel extensions: An ongoing debate," in UML Workshop 2000,* California, 2000.

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

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

[33] T. Gruber, *"Toward Principles for the Design of Ontologies Used for Knowledge Sharing," in International Workshop on Formal Ontology*, Italy, 1993.

[34] M. Karlsson, L. Olsson, *"pRTI 1516 - Rationale and Design," in 2001 Fall Simulation Interoperability Workshop (SIW),* Orlando, 2001.

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

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

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

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

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

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

[41] S. Parr, *"A Visual Tool to Simplify the Building of Distributed Simulations Using HLA," in Information & Security International Journal,* vol. 12 no. 2, pp. 151-163, 2003.

[42] T. Rathnam, C.J.J. Paredis, *"Developing Federation Object Models Using Ontologies," in the Proceedings of the 2004 Winter Simulation Conference,* Washington D.C., 2004.

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

[44] O. Topçu, H. Oğuztüzün, *"Towards a UML Extension for HLA Federation Design," in the Proceedings of 2nd Conference on Simulation Methods and Applications (CSMA-2000),* Orlando, 2000.

[45] O. Topçu, H. Oğuztüzün, G.M. Hazen, *"Towards a UML Profile for HLA Federation Design, Part II," in the Proceedings of Summer Computer Simulation Conference (SCSC-2003),* Montreal, Canada, 2003.

[46] A. Vizhanyo, *"A Metamodel based Model Transformation Tool: GReAT and Code Generator: the GReAT Compiler," in OMG MIC Workshop,* Virginia, 2004.

APPENDIX A

USER'S GUIDE

This is the User's Guide of the Object Model Design Environment (OMDE) which is automatically generated with GME. It explains how to use the GME HLA OM paradigm to design object models.

Registering the paradigm:

To be able to use the OMDE you should first register the HLA OM paradigm. Open the HLA OM paradigm ".mga" file and run the Meta 2004 Interpreter on the upper left corner of the GME. Answer "yes" when you are asked to register the paradigm. And press "Next" button for other dialogs. Close the metamodel.

Opening a New HLA OMT Project:

Start GME, and select "File/New Project". A dialog box asks you to choose the paradigm that the new project will be based on. Select HLA_OMT and press the "Create New" button. The next dialog asks you to specify the data storage. Simple models are usually stored in project files. Click "Next" and you are asked to name a project file. The standard extension is ".mga". Specify a name (like "restaurant.mga") and press OK. GME has now created and opened an empty project that is named "restaurant" and associated with the HLA_OMT paradigm.

Creating Federation Design model:

Right click on the root folder in the Browser window (the one usually positioned at the right side), and select the single option "Federation Design Model" within the "Insert Model" option as in Figure 4.20. A new model named "New Federation Design Model" is created under the root; you may change the name from Attributes browser. Double click on the model to open it. An empty window appears in the user-area.

The Part Browser, a small window in the lower left portion of the program, displays the model elements that can be inserted into the model in its current aspect. The elements in this browser are federation, federate, FOMReference and SOMReference. You can use them by dragging from the Part Browser onto the main window. You can connect federation to federates to denote the members of the federation; federation to FOMReference; and Federate to SOMReference. When using references, you drag the referred element over the reference and drop it when the mouse icon changes. But before referring elements you should first define FOM and SOM object models. Copy and paste operations on elements are supported by GME and all elements can be created, moved or copied by drag and drop as usual.

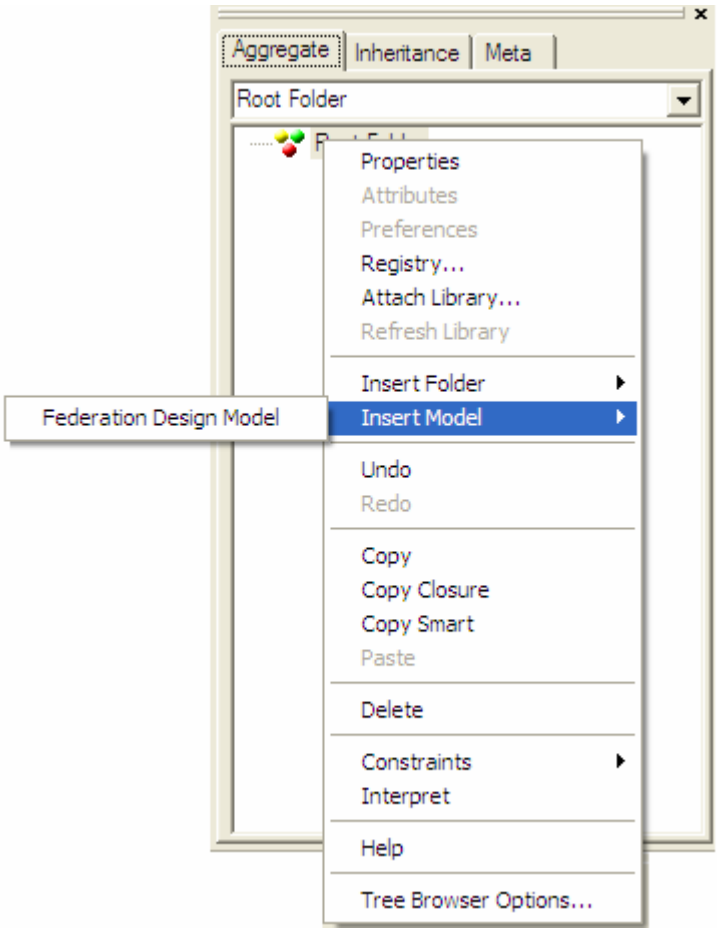

Figure 4.20: Creating Federation Design model.

Creating Object Model:

Right click on the root folder in the Browser window and select the "Object Models" within the "Insert Folder" option. A new folder named "New Object Models" is created under the root. Again right click on the new folder. You can select FOM, SOM or Other within the "Insert Model" option. For example select "SOM". Double click to open it. An empty window appears in the user-area. Figure 4.21 shows a part of the Restaurant SOM.

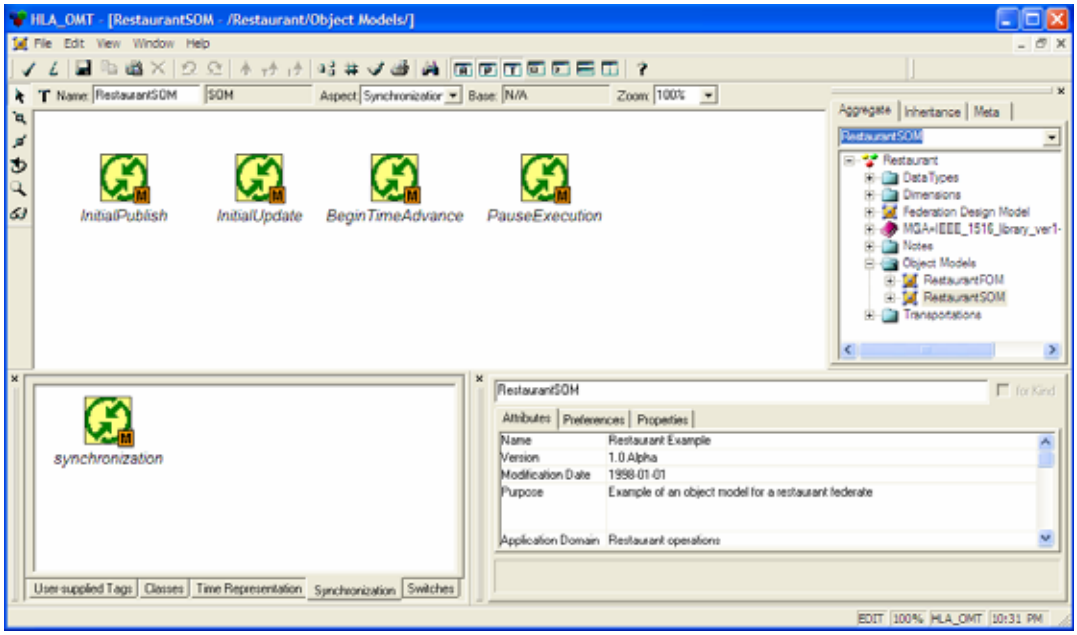

Figure 4.21: Creating Object Models.

The Part Browser has five views and each view displays the model elements that can be inserted into the model in its current aspect.

- Classes-view includes the definition of object classes, attributes, interaction classes, and the inheritance relation between the classes.
- User-Supplied Tags view includes user supplied tag elements.
- Synchronization view includes the definition of synchronization point models.
- Switches view includes the Boolean switches in order to change the initial settings

- Time representation view includes the definition of lookahead and timestamp models.

The icons with “M” letter on the bottom mean this element has parts; you can open the parts double clicking on the icon.

Creating Other Elements:

Right click on the root folder in the Browser window within the “Insert Folder” option you can select Dimensions, Transportations, Data Types or Notes. After the selection a new folder is created under the root. By right clicking on the new folder, you can select the related model elements. And in the same way with object models and federation design model, you can define the model elements.

For adding HLA notes, define notes under the Notes folder and give references to them; or directly add notes to the notes attribute of each modeling element. For adding design notes, use Annotation facility of GME.

And lastly we can add a few points. References used in the model shall not be null, if you want to denote “NA”, then simply use no reference element. If you want to check the validity of your model you can use Check facility of GME, by selecting “File-> Check” option.

APPENDIX B

IEEE DEFAULT LIBRARY

This is the XME output of IEEE Default Datatypes and Transportations Library defined with object model design environment.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE project SYSTEM "mga.dtd">

<project version="IEEE 1516.2-2000" metaname="HLA_OMT">
  <name>IEEE 1516 defaults</name>
  <comment>Default entries for HLA-OMT.</comment>
  <author>Deniz Cetinkaya</author>

  <folder id="id-006a-00000001" kind="RootFolder">
    <name>IEEE 1516 defaults</name>
    <folder id="id-006a-00000002" kind="dataTypes">
      <name>Default Data Types</name>
      <folder id="id-006a-00000003" kind="basicDataRepr">
        <name>Default Basic Data Representations</name>
        <model id="id-0065-00000001" kind="basicData">
          <name>HLAfloat32BE</name>
          <attribute kind="encoding">
            <value>32-bit IEEE normalized single-precision
            format.
            (see IEEE Std. 754-1985)</value>
          </attribute>
          <attribute kind="endian" status="meta">
            <value>Big</value>
          </attribute>
          <attribute kind="interpretation">
            <value>Single-precision floating-point number.
            </value>
          </attribute>
          <attribute kind="notes" status="meta">
            <value></value>
          </attribute>
          <attribute kind="size">
            <value>32</value>
          </attribute>
        </model>
        <model id="id-0065-00000002" kind="basicData">
          <name>HLAfloat32LE</name>
          <attribute kind="encoding">
            <value>32-bit IEEE normalized single-precision
            format.
```

```
    (see IEEE Std. 754-1985)</value>
  </attribute>
  <attribute kind="endian">
    <value>Little</value>
  </attribute>
  <attribute kind="interpretation">
    <value>Single-precision floating-point number.
    </value>
  </attribute>
  <attribute kind="notes" status="meta">
    <value></value>
  </attribute>
  <attribute kind="size">
    <value>32</value>
  </attribute>
</model>
<model id="id-0065-00000003" kind="basicData">
  <name>HLAfloat64BE</name>
  <attribute kind="encoding">
    <value>64-bit IEEE normalized single-precision
    format.
    (see IEEE Std. 754-1985)</value>
  </attribute>
  <attribute kind="endian" status="meta">
    <value>Big</value>
  </attribute>
  <attribute kind="interpretation">
    <value>Double-precision floating-point number.
    </value>
  </attribute>
  <attribute kind="notes" status="meta">
    <value></value>
  </attribute>
  <attribute kind="size">
    <value>64</value>
  </attribute>
</model>
<model id="id-0065-00000004" kind="basicData">
  <name>HLAfloat64LE</name>
  <attribute kind="encoding">
    <value>64-bit IEEE normalized single-precision
    format.
    (see IEEE Std. 754-1985)</value>
  </attribute>
  <attribute kind="endian">
    <value>Little</value>
  </attribute>
  <attribute kind="interpretation">
    <value>Double-precision floating-point number.
    </value>
  </attribute>
  <attribute kind="notes" status="meta">
    <value></value>
  </attribute>
  <attribute kind="size">
    <value>64</value>
  </attribute>
</model>
```

```
<model id="id-0065-00000005" kind="basicData">
  <name>HLAinteger16BE</name>
  <attribute kind="encoding">
    <value>16-bit two&amp;apos;s complement signed
    integer.
    The most significant bit contains the sign.
    </value>
  </attribute>
  <attribute kind="endian" status="meta">
    <value>Big</value>
  </attribute>
  <attribute kind="interpretation">
    <value>Integer in the range [-2^15, -2^15-1].
    </value>
  </attribute>
  <attribute kind="notes" status="meta">
    <value></value>
  </attribute>
  <attribute kind="size">
    <value>16</value>
  </attribute>
</model>
<model id="id-0065-00000006" kind="basicData">
  <name>HLAinteger16LE</name>
  <attribute kind="encoding">
    <value>16-bit two&amp;apos;s complement signed
    integer.
    The most significant bit contains the sign.
    </value>
  </attribute>
  <attribute kind="endian">
    <value>Little</value>
  </attribute>
  <attribute kind="interpretation">
    <value>Integer in the range [-2^15, -2^15-1].
    </value>
  </attribute>
  <attribute kind="notes" status="meta">
    <value></value>
  </attribute>
  <attribute kind="size">
    <value>16</value>
  </attribute>
</model>
<model id="id-0065-00000007" kind="basicData">
  <name>HLAinteger32BE</name>
  <attribute kind="encoding">
    <value>32-bit two&amp;apos;s complement signed
    integer.
    The most significant bit contains the sign.
    </value>
  </attribute>
  <attribute kind="endian" status="meta">
    <value>Big</value>
  </attribute>
  <attribute kind="interpretation">
    <value>Integer in the range [-2^31, -2^31-1].
    </value>
```

```
  </attribute>
  <attribute kind="notes" status="meta">
    <value></value>
  </attribute>
  <attribute kind="size">
    <value>32</value>
  </attribute>
</model>
<model id="id-0065-00000008" kind="basicData">
  <name>HLAinteger32LE</name>
  <attribute kind="encoding">
    <value>32-bit two&amp;apos;s complement signed
    integer.
    The most significant bit contains the sign.
    </value>
  </attribute>
  <attribute kind="endian">
    <value>Little</value>
  </attribute>
  <attribute kind="interpretation">
    <value>Integer in the range [-2^31, -2^31-1].
    </value>
  </attribute>
  <attribute kind="notes" status="meta">
    <value></value>
  </attribute>
  <attribute kind="size">
    <value>32</value>
  </attribute>
</model>
<model id="id-0065-00000009" kind="basicData">
  <name>HLAinteger64BE</name>
  <attribute kind="encoding">
    <value>64-bit two&amp;apos;s complement signed
    integer.
    The most significant bit contains the sign.
    </value>
  </attribute>
  <attribute kind="endian" status="meta">
    <value>Big</value>
  </attribute>
  <attribute kind="interpretation">
    <value>Integer in the range [-2^63, -2^63-1].
    </value>
  </attribute>
  <attribute kind="notes" status="meta">
    <value></value>
  </attribute>
  <attribute kind="size">
    <value>64</value>
  </attribute>
</model>
<model id="id-0065-0000000a" kind="basicData">
  <name>HLAinteger64LE</name>
  <attribute kind="encoding">
    <value>64-bit two&amp;apos;s complement signed
    integer.
    The most significant bit contains the sign.
```

```
    </value>
  </attribute>
  <attribute kind="endian">
    <value>Little</value>
  </attribute>
  <attribute kind="interpretation">
    <value>Integer in the range [-2^63, -2^63-1].
    </value>
  </attribute>
  <attribute kind="notes" status="meta">
    <value></value>
  </attribute>
  <attribute kind="size">
    <value>64</value>
  </attribute>
</model>
<model id="id-0065-0000000b" kind="basicData">
  <name>HLAoctet</name>
  <attribute kind="encoding">
    <value>Assumed to be portable among hardware
    devices.</value>
  </attribute>
  <attribute kind="endian" status="meta">
    <value>Big</value>
  </attribute>
  <attribute kind="interpretation">
    <value>8-bit value</value>
  </attribute>
  <attribute kind="notes" status="meta">
    <value></value>
  </attribute>
  <attribute kind="size">
    <value>8</value>
  </attribute>
</model>
<model id="id-0065-0000000c" kind="basicData">
  <name>HLAoctetPairBE</name>
  <attribute kind="encoding">
    <value>Assumed to be portable among hardware
    devices.</value>
  </attribute>
  <attribute kind="endian" status="meta">
    <value>Big</value>
  </attribute>
  <attribute kind="interpretation">
    <value>16-bit value</value>
  </attribute>
  <attribute kind="notes" status="meta">
    <value></value>
  </attribute>
  <attribute kind="size">
    <value>16</value>
  </attribute>
</model>
<model id="id-0065-0000000d" kind="basicData">
  <name>HLAoctetPairLE</name>
  <attribute kind="encoding">
    <value>Assumed to be portable among hardware
```

```
            devices.</value>
          </attribute>
          <attribute kind="endian">
            <value>Little</value>
          </attribute>
          <attribute kind="interpretation">
            <value>16-bit value</value>
          </attribute>
          <attribute kind="notes" status="meta">
            <value></value>
          </attribute>
          <attribute kind="size">
            <value>16</value>
          </attribute>
        </model>
      </folder>
      <folder                          id="id-006a-00000004"
kind="enumeratedDatas">
        <name>Default Enumerated Data Types</name>
        <model                         id="id-0065-0000000e"
kind="enumeratedData">
          <name>HLAboolean</name>
          <attribute kind="notes" status="meta">
            <value></value>
          </attribute>
          <attribute kind="semantics">
            <value>Standard Boolean type</value>
          </attribute>
          <atom id="id-0066-00000001" kind="enumerator">
            <name>HLAfalse</name>
            <attribute kind="notes" status="meta">
              <value></value>
            </attribute>
            <attribute kind="values">
              <value>0</value>
            </attribute>
          </atom>
          <atom id="id-0066-00000002" kind="enumerator">
            <name>HLAtrue</name>
            <attribute kind="notes" status="meta">
              <value></value>
            </attribute>
            <attribute kind="values">
              <value>1</value>
            </attribute>
          </atom>
          <reference id="id-0067-00000001"
         kind="representation"           referred="id-0065-
00000007">
            <name>representation</name>
          </reference>
        </model>
      </folder>
      <folder id="id-006a-00000005" kind="arrayDataTypes">
        <name>Default Array Data Types</name>
        <model id="id-0065-0000000f" kind="arrayData">
          <name>HLAASCIIstring</name>
          <attribute kind="cardinality" status="meta">
```

```
            <value>Dynamic</value>
          </attribute>
          <attribute kind="encoding">
            <value>HLAvariableArray</value>
          </attribute>
          <attribute kind="notes" status="meta">
            <value></value>
          </attribute>
          <attribute kind="semantics">
            <value>ASCII string representation</value>
          </attribute>
          <reference id="id-0067-00000005" kind="datatype"
          referred="id-0065-00000012">
            <name>datatype</name>
          </reference>
        </model>
        <model id="id-0065-00000010" kind="arrayData">
          <name>HLAopaqueData</name>
          <attribute kind="cardinality" status="meta">
            <value>Dynamic</value>
          </attribute>
          <attribute kind="encoding">
            <value>HLAvariableArray</value>
          </attribute>
          <attribute kind="notes" status="meta">
            <value></value>
          </attribute>
          <attribute kind="semantics">
            <value>Uninterpreted sequence of bytes</value>
          </attribute>
          <reference id="id-0067-00000007" kind="datatype"
          referred="id-0065-00000014">
            <name>datatype</name>
          </reference>
        </model>
        <model id="id-0065-00000011" kind="arrayData">
          <name>HLAunicodeString</name>
          <attribute kind="cardinality" status="meta">
            <value>Dynamic</value>
          </attribute>
          <attribute kind="encoding">
            <value>HLAvariableArray</value>
          </attribute>
          <attribute kind="notes" status="meta">
            <value></value>
          </attribute>
          <attribute kind="semantics">
            <value>Unicode string representation</value>
          </attribute>
          <reference id="id-0067-00000008" kind="datatype"
          referred="id-0065-00000013">
            <name>datatype</name>
          </reference>
        </model>
      </folder>
      <folder                          id="id-006a-00000006"
kind="simpleDataTypes">
        <name>Default Simple Data Types</name>
```

```
        <model id="id-0065-00000012" kind="simpleData">
          <name>HLAASCIIchar</name>
          <attribute kind="accuracy" status="meta">
            <value>NA</value>
          </attribute>
          <attribute kind="notes" status="meta">
            <value></value>
          </attribute>
          <attribute kind="resolution" status="meta">
            <value>NA</value>
          </attribute>
          <attribute kind="semantics">
            <value>Standard ASCII character.
            (See ANSI Std. X3.4-1986)</value>
          </attribute>
          <attribute kind="units" status="meta">
            <value>NA</value>
          </attribute>
          <reference id="id-0067-00000002"
         kind="representation"           referred="id-0065-
0000000b">
            <name>representation</name>
          </reference>
        </model>
        <model id="id-0065-00000013" kind="simpleData">
          <name>HLAunicodeChar</name>
          <attribute kind="accuracy" status="meta">
            <value>NA</value>
          </attribute>
          <attribute kind="notes" status="meta">
            <value></value>
          </attribute>
          <attribute kind="resolution" status="meta">
            <value>NA</value>
          </attribute>
          <attribute kind="semantics">
            <value>Unicode UTF-16 character.
            (see    the    Unicode    Standard,    version
3.0)</value>
          </attribute>
          <attribute kind="units" status="meta">
            <value>NA</value>
          </attribute>
          <reference id="id-0067-00000003"
         kind="representation"           referred="id-0065-
0000000c">
            <name>representation</name>
          </reference>
        </model>
        <model id="id-0065-00000014" kind="simpleData">
          <name>HLAbyte</name>
          <attribute kind="accuracy" status="meta">
            <value>NA</value>
          </attribute>
          <attribute kind="notes" status="meta">
            <value></value>
          </attribute>
          <attribute kind="resolution" status="meta">
```

```
            <value>NA</value>
          </attribute>
          <attribute kind="semantics">
            <value>Uninterpreted 8-bit value.</value>
          </attribute>
          <attribute kind="units" status="meta">
            <value>NA</value>
          </attribute>
          <reference id="id-0067-00000004"
         kind="representation"          referred="id-0065-
0000000b">
            <name>representation</name>
          </reference>
        </model>
      </folder>
    </folder>
    <folder id="id-006a-00000007" kind="transportations">
      <name>Default Transportations</name>
      <atom id="id-0066-00000003" kind="transportation">
        <name>HLAreliable</name>
        <attribute kind="description">
          <value>Provide reliable delivery of data in the
          sense
          that TCP/IP delivers its data reliably.</value>
        </attribute>
        <attribute kind="notes" status="meta">
          <value></value>
        </attribute>
      </atom>
      <atom id="id-0066-00000004" kind="transportation">
        <name>HLAbestEffort</name>
        <attribute kind="description">
          <value>Make an effort to deliver data in the
          sense
          that UDP provides best-effort delivery.</value>
        </attribute>
        <attribute kind="notes" status="meta">
          <value></value>
        </attribute>
      </atom>
    </folder>
  </folder>
</project>
```