ONTOLOGY DRIVEN DEVELOPMENT FOR HLA FEDERATES

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

BY

CEREN FATMA KÖKSAL ALGIN

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

APRIL 2010

Approval of the thesis:

ONTOLOGY DRIVEN DEVELOPMENT FOR HLA FEDERATES

submitted by **CEREN FATMA KÖKSAL ALGIN** in partial fulfillment of the requirements for the degree of **Master in Computer Engineering Department, Middle East Technical University** by,

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

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

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

Dr. Umut Durak
Co-Supervisor, **TÜBİTAK, SAGE** _____

Examining Committee Members:

Prof. Dr. Adnan Yazıcı
Computer Engineering Dept., METU _____

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

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Dept., METU _____

Asst. Prof. Dr. Pınar Şenkul
Computer Engineering Dept., METU _____

Dr. Umut Durak
TÜBİTAK, SAGE _____

Date: 30.04.2010 _____

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: Ceren Fatma KÖKSAL ALGIN

Signature :

ABSTRACT

ONTOLOGY DRIVEN DEVELOPMENT FOR HLA FEDERATES

Köksal Algın, Ceren Fatma
M.Sc., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Halit Oğuztüzün
Co-Supervisor: Dr. Umut Durak

April 2010, 66 pages

This thesis puts forth a process for ontology driven distributed simulation through a case study. Ontology is regarded as a domain model, including objects, attributes, methods and object relations. The case study involves trajectory simulation. A trajectory simulation is a piece of software that calculates the flight path and other parameters of a munition, such as its orientation and angular rates, from launch to impact. Formal specification of trajectory simulation domain is available as a domain model in the form of an ontology, called Trajectory Simulation ONTology (TSONT). Ontology driven federation development process proposed in this thesis is executed in three steps. The first step is to analyze the TSONT and to create instances of individuals guided by the requirements of the targeted simulation application, called Puma Trajectory Simulation. Puma is the simulation of a ficticious air-to-ground guided bomb. The second step is to create the High Level Architecture(HLA) Federation Object Model (FOM) using Puma Simulation individuals. FOM will include the required object and interaction definitions to enable information exchange among federation members, including the Puma federate and the Exercise Manager federate. Transformation from the ontology to FOM is realized in two ways: manually, and by using a tool called OWL2OMT. The third step is to implement the Trajectory Simulation federation based on the constructed FOM. Thus, the applicability of developing HLA federates and the federation under the guidance of ontology is demonstrated.

Keywords: Trajectory Simulation Ontology, High Level Architecture, Federation Object Model, Interoperability

ÖZ

YÜKSEK SEVİYE MİMARİ FEDERELERİ İÇİN ONTOLOJİYE DAYALI GELİŞTİRME

Köksal Algın, Ceren Fatma
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Halit Oğuztüzün
Ortak Tez Yöneticisi: Dr. Umut Durak

Nisan 2010, 66 sayfa

Bu tez, ontolojiye dayalı dağıtık benzetim geliştirmek için örnek bir olay incelemesi yoluyla bir süreç ortaya koymaktadır. Ontoloji, nesneler, nitelikler, yöntemler ve nesneler arası ilişkileri içeren bir ilgi alanı modelidir. Buradaki örnek olay incelemesi yörünge benzetimidir. Yörünge benzetimi, bir mühimmata ait uçuş rotasını, fırlatılmadan etki anına kadar, ilgili parametreleriyle hesaplayan bir yazılım parçasıdır. Kurallarla tanımlanmış yörünge benzetim ilgi alan modeli, bir ontoloji biçiminde Yörünge Benzetim Ontolojisi (TSONT) adıyla hazır bulunmaktadır. Ontolojiye dayalı federasyon geliştirme süreci, bu tezde üç adımda gerçekleştirilecektir. İlk adım TSONT'u analiz etmek ve "Puma Yörünge Benzetimi" adındaki hedef benzetim uygulamasının gereksinimleriyle yönlendirerek, ontolojiye ait fertlerin benzetim örneklerini yaratmak olacaktır. Puma, kurgusal bir havadan yere güdümlü bombanın benzetimidir. İkinci adım Puma Benzetimi fertlerine ait Yüksek Seviye Mimari(HLA) standardında Federasyon Nesne Modeli(FOM) oluşturmaktır. FOM, nesne ve etkileşim tanımları gibi gerekli olan bilgileri içererek, Puma federe ve Exercise Manager federe gibi federasyon üyeleri arasında bilginin paylaşılabilmesini sağlar. Ontolojiden FOM'a geçiş iki yolla uygulanacaktır: elle ve OWL2OMT adında bir araç kullanılarak. Üçüncü adım ise Yörünge Benzetim Federasyonunu üretilmiş olan FOM'a dayalı olarak geliştirmek olacaktır. Böylece, HLA federeleri ve federasyonunun ontolojinin rehberliği ile geliştirilmesinin uygulanabilirliği gösterilmiş olacaktır.

Anahtar Kelimeler: Yörünge Benzetimi Ontolojisi, Yüksek Seviye Mimari, Federasyon Nesne Modeli, Birlikte işlerlik

To My Family

ACKNOWLEDGEMENTS

I especially thank to my supervisors, Assoc. Prof. Dr. Halit Oğuztüzün for supervising and guiding me, providing resources, subjects, and for also offering direction and insight throughout the research, and to Dr. Umut Durak for his valuable supervision, suggestions and comments during my thesis.

I would like to thank to collaborative work with the Özer Özdikiş, who developed the OWL2UML and OWL2OMT tools.

I also thank Havelsan Inc. for providing a supportive environment to study for the degree of master.

Finally, I would like to thank my husband, my brother and my parents, for their support against all the difficulties that I met.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

FOM	Federation Object Model
HLA	High Level Architecture
IEEE	Institute of Electrical and Electronic Engineers
METU	Middle East Technical University
OMT	Object Model Template
OWL	Web Ontology Language
pRTI	Portable Runtime Infrastructure
RTI	Runtime Infrastructure
SOM	Simulation Object Model
TSONT	Trajectory Simulation Ontology
UML	Unified Modeling Language

CHAPTER 1

INTRODUCTION

This chapter introduces the motivation and scope of the study, summarizes the path from ontology to HLA and then outlines the organization of the thesis.

1.1 Motivation and Scope

We propose a process for developing a kind of trajectory simulation which begins at TSONT and ends at an HLA federate. Trajectory simulation, in the present context, means computing the flight path and other parameters, such as orientation, and angular rates, of the munition from the start to the end of its motion [1]. Munition's and its subsystems' behavior during the flight are subjects of Trajectory simulation domain. Mathematical models are constructed and then computed to determine the acceleration, velocity, position and attitude of the munition.

Several trajectory simulations have been developed for different purposes and usage areas. Their performance and characteristics differ widely from each other. Trajectory simulation ontology comprises the fundamental knowledge in trajectory simulation domain and makes it available for human understanding and machine processing. The process we define for developing trajectory simulations aims to enhance interoperability and composability. Interoperability and composability are critical issues in distributed simulation development. While interoperability can be defined as the capability of simulations in exchanging information [2], composability is defined as the capability that enables selecting and assembling components in various combinations [3].

Main objective in this thesis is to investigate the applicability of an ontology driven federate development process through a case study in which a Federation Object Model (FOM) is derived from TSONT. We demonstrate the process through an example of an air to ground guided bomb simulation. First and foremost, TSONT has been analyzed according to the requirements of the simulation. Individuals which will be the facilitators of developing the FOM were created in TSONT.

Properties of the individuals are examined to decide the level of detail which will be important during Federation Object Model generation.

FOM is constructed in two ways, first manually and then by using the OWL2OMT tool. Puma Trajectory Simulation is the module which computes the trajectory of the guided bomb. An HLA wrapper is developed to incorporate the Puma Trajectory Simulation into the application. Using FOM, Puma Federate code is written in accordance with pRTI's library [24] and HLA IEEE 1516.1-2000 standard. Thus, Puma federate that calls a 6DOF trajectory simulation and Exercise Manager federate that tracks the trajectory information updates, were developed and made available to join any federation that requires trajectory simulation of munitions. In this case study the federation is called Trajectory Simulation federation.

1.2 TSONT to HLA

Ontology based trajectory simulation is the output of the transition from TSONT to HLA OMT. This idea was first presented at [4]. Ontology Based Trajectory Reuse Infrastructure aimed at enabling reuse from knowledge to code. In this work we utilized the practices presented in the Ontology Based Trajectory Reuse Infrastructure for developing an interoperable and composable trajectory simulation that will be available for reuse. Ontology based trajectory simulation development process is presented in Figure 1. This figure consists of two transformations. Upper part starts with Web Ontology Language (OWL) to OMT transformation and the lower part starts with OWL to UML transformation.

Platform Independent Framework Architecture is constructed by using model transformation supported by a tool named OWL2UML [5]. This is the UML related phase of the trajectory simulation. OWL2UML enables user guided transformations from OWL ontology to a UML class diagram. Then Platform Independent Framework Architecture derives the MATSIX – MATLAB 6DOF Trajectory Simulation Framework which implements the architecture.

"Framework architecture is designed specifically for MATLAB six Degrees Of Freedom (6DOF) simulations. MATSIX, the simulation framework is implemented in MATLAB. In this effort we utilized MATSIX for computing the trajectory of the guided bomb"[25].

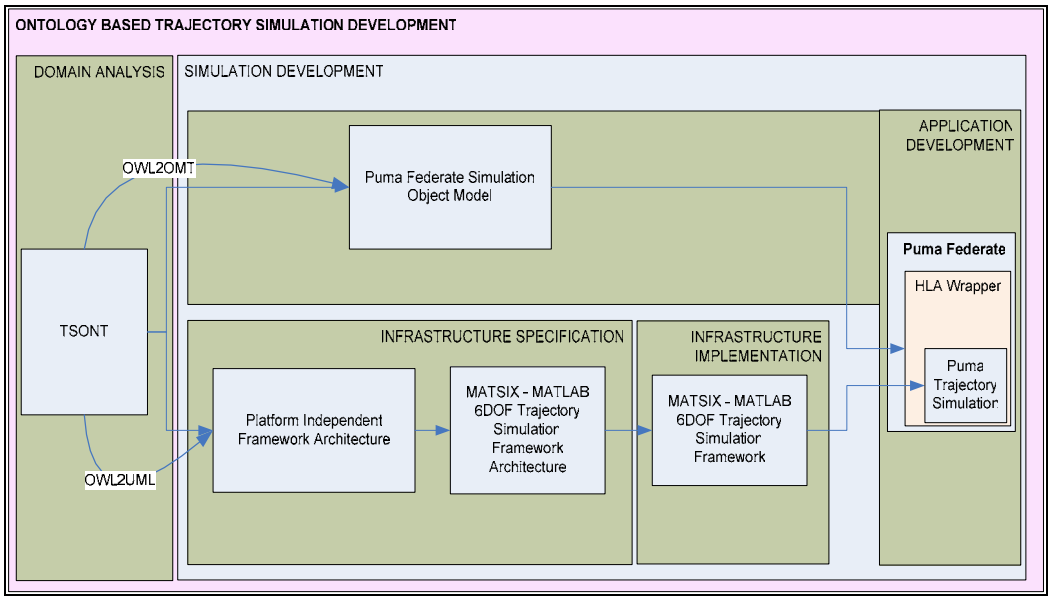


Figure 1 Ontology Based Trajectory Simulation Development

Trajectory simulation domain model is transformed to OMT as a FOM for Trajectory Simulation Federation. Federation Object Model is composed of the objects, their attributes and the interactions, their parameters which will be used in the federation to achieve interoperability. Conversion from the domain model to the Federation Object Model is substantiated in two approaches: The first one is manually analyzing the domain and the requirements of the simulation and writing the rules of FOM. The second one is converting by a tool named OWL2OMT. The OWL2OMT tool enables user guided extraction of Simulation Object Models from OWL ontology [6]. By using and obeying the HLA OMT, Puma Federate and Exercise Manager Federate (which tracks the trajectory) are implemented in Java programming language. This is the HLA wrapper for Puma Trajectory Simulation. The MATLAB executable which is mentioned in the paragraph before is wrapped with Java code that links to Runtime Infrastructures API.

This thesis benefits from some practices reported in the thesis by Umut Durak [11] which were formerly completed. Analyzing the domain and composing the FOM, implementing the federates and the federation based on the FOM are the originally developed parts of the present thesis.

1.3 Organization of the Thesis

The preceding sections of this chapter introduce the motivation and scope of the study and present the summary of the thesis application. The remaining chapters are broken down as follows:

- Chapter 2 provides related literature and background information required to easily understand the subsequent chapters.
- Chapter 3 explains how to generate models of the exchanged objects from the trajectory simulation ontology TSONT.
- Chapter 4 describes developing the application, namely federates which wrap Puma Trajectory Simulation.
- Finally, Chapter 5 discusses the accomplishments and draws conclusions.

CHAPTER 2

BACKGROUND

In this chapter, the background information is summarized from the related literature which guides the reader to follow the details in the subsequent chapters. First, usage of ontology in Modeling & Simulation up to present is reviewed. Second, Trajectory Simulation ONTology, the domain model of the Puma Trajectory Simulation, is discussed. Third, Puma Trajectory Simulation is outlined. Fourth, High Level Architecture, which is the standard Puma Trajectory Simulation conforms to is introduced. Finally, HLA Programming in Java is described with its rules and primary methods which constitute the HLA wrapper code for Puma.

2.1 Usage of Ontology in Modeling & Simulation

Before discussing the usage of ontology in modeling and simulation, we define what ontology is. Neches gives a description of ontology as "An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary"[7].

Ontology which Trajectory Simulation Ontology is based on is OWL. "OWL, Web Ontology Language, is designed for use by applications that need to process the content of information. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics."[http://www.w3.org/2004/OWL/]

Importance of using Ontology in Modeling and Simulation has been recognized in simulation world. If we glance at Turnitsa and Tolk their observation is, "Ontology produces a representation of the world that a machine can process. In the case of most application domains, that world can be tested and verified outside of the system – it is just a representation. However, in the field of simulations, the synthetic environment of a simulation has reality only within the system. In the case of a simulation, the ontology doesn't just serve to describe the representation of the world; it serves to describe the

(simulated) world itself. Appreciating this leads us to see how ontological representation is important for design, validation, interoperability and other tasks." [16].

An ontological description of a domain space can make it much easier for design, architecture, verification, validation, and interoperation of systems within that domain [17]. These are valuable goals, and certainly of great interest to the modeling and simulation community [16].

Usage of ontology in modeling and simulation can be divided into two categories referring to Tolk and Turnitsa: Static information models, and dynamic information systems.

In Static Information Models, static ontological representation is a key part. C-BML, MSDL and SEDRIS are examples to static information models.

C-BML (Coalition Battle Management Language) is a project that has the aim of unambiguously conveying information between a C2 system as a source, and either another C2 system, M&S system or robotic system as a target. Interest to us here is the connection between meaning and terms (or between Doctrine and Representation), which has been identified as the Ontology Layer [18].

MSDL (Military Scenario Definition Language) is an effort to describe all of the aspects that are part of a military scenario [19]. This includes the environment, entities, orders, and effects. Migrating to either the taxonomy or ontology strata of the spectrum would require basing the elements of MSDL on concepts, rather than just terms [16].

SEDRIS (the Synthetic Environment Data Representation and Interchange Specification) has been the object of interest of a group that is representing the domain ontological meaning that is shown through the SEDRIS data representation formats. This group has presented a view of how an ontology could be developed to support making that ontological meaning known and portable between applications [20]. Relying on this research, Turnitsa and Tolk believe that as more applications can make use of ontological data during their exchange, either a Neutral Authoring or Common Access to Information method might arise [16].

The second category is Dynamic Information Systems. One of the aspects of M&S that makes applying ontological representation both crucial, as well as difficult, is that a simulated environment is a dynamic world. In order to support such a dynamic domain view, an ontological representation must be able to adapt. It should be based on concepts, as they are universal and won't frequently change, yet the relations between various terms that those concepts give meaning to will shift as time changes within the simulated environment [16].

There is also another usage of ontology in modeling as the subject of some other papers. DeMO(Discrete-event Modeling Ontology) is a Web-accessible ontology for discrete-event modeling (DEM). DeMO has four main abstract classes representing main concepts in the knowledge domain: DeModel, ModelConcepts, Model-Components and ModelMechanisms [21].

DeMO specifies the knowledge concepts about different existing formalisms and the relations between those concepts. Any discrete-event modeling formalism may, in principle, be inserted in this repository and the relations to other formalisms and the underlying concepts may be represented as well. Making ontology accessible from Web is an efficient way for modelers and computer applications to utilize and follow up that knowledge repository. The presence of this explicit knowledge on the Web organized in such a fashion is regarded as a useful step in the evolution of Web-based modeling and simulation [22]. Ontology driven model development and use, specific for DeMO can be seen in Figure 2.

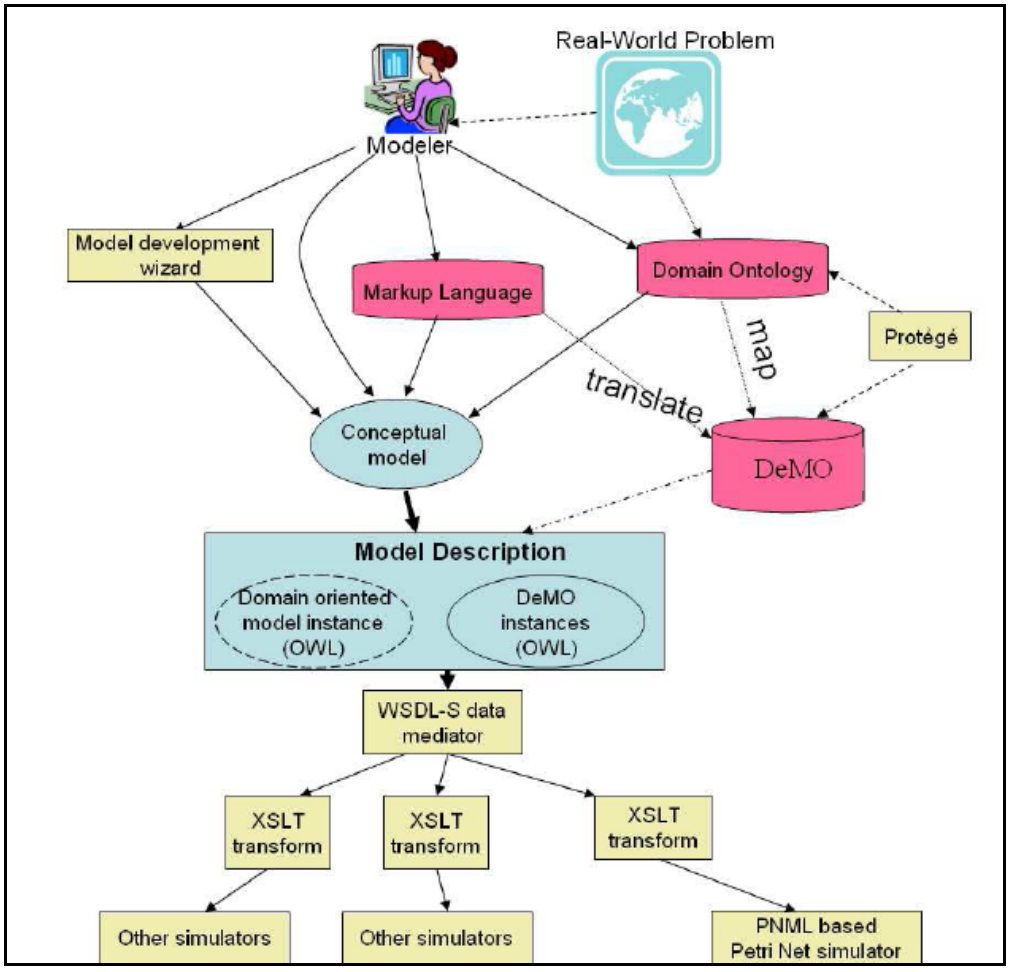


Figure 2 Ontology driven model development and use [22]

OWL is also beginning to be used in the United States in military applications such as:

- Joint Explosive Ordnance Disposal (JEOD) Decision Support System (DSS)
- Foreign Clearance Guide (FCG)
- Computer Generated Forces (CFG) Human Behavior Representation (HBR) [23]

There is currently no real agreement over whether a dynamic domain would be best served by an ontological representation that is large enough to encompass all of the changes within a static structure, or if the ontological representation would have to be dynamic. In the present thesis, our position is to employ a static structure which is large enough to encompass all of the changes in the domain.

2.2 Trajectory Simulation ONTology (TSONT)

Following Gruber [8], knowledge in ontologies can be defined with the components: concepts, relations, functions, axioms and instances. Some of these components will be presented specifically for TSONT in the further paragraphs.

The basic idea behind developing an ontology as the domain model of the trajectory simulation domain is first to establish a common vocabulary that is agreed among people working on trajectory simulations. Another main consideration is to create a backbone for systematization of knowledge on how to build a trajectory simulation [9].Potential benefits of this approach include documentation, maintenance, reliability, knowledge reuse and interoperability of the developed applications [10].

Engineering knowledge, used to simulate the trajectory of a munition from launch to impact is stored in an ontology called Trajectory Simulation ONTology (TSONT). Concepts of trajectory simulation and the relation among these concepts are captured by using Web Ontology Language and presented as a domain model that is available for reuse [11].

2.2.1 Overview of TSONT

As we mentioned earlier TSONT is a domain model which contains the information to simulate a munition's flight path from leaving a launcher till engaging a target. TSONT was developed by Durak as a part of his Ph.D. thesis "Ontology Based Reuse Infrastructure For Trajectory Simulation" [11]. Elements of TSONT can be outlined as attributes, classes, composite data, functions, objects and quantities as seen in Figure 3.

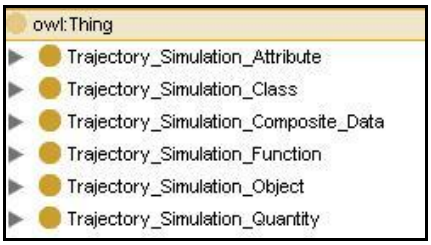


Figure 3 TSONT Domain Model Elements

2.2.2 Trajectory Simulation Objects

Trajectory Simulation Objects can be readily described in physical terms. Munition, munition subsystem and weapon are the elements of the Trajectory Simulation Objects as seen in Figure 4. Munition is comprised of ammunition, bomb and missile. The munition which is simulated in this thesis is a type of Guided bomb, which is under Bomb in the hierarchy.

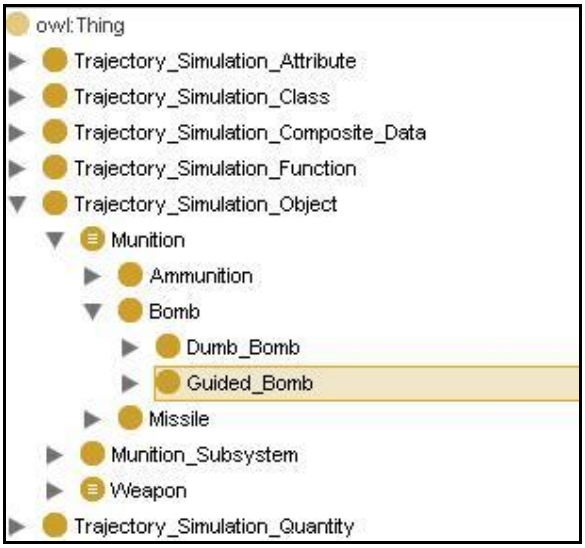


Figure 4 Trajectory Simulation Objects

2.2.3 Trajectory Simulation Classes

A class is defined as an implicit collection of individuals that belong together because they share some properties. Coordinate System, Model, Parameter, Solver, Trajectory Simulation and Trajectory Simulation Phase are the classes of the TSONT domain as seen in Figure 5.

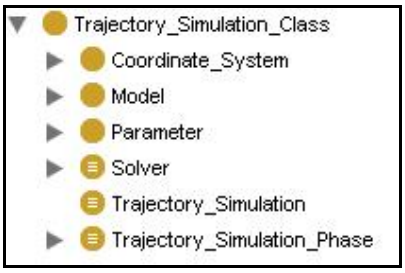


Figure 5 Trajectory Simulation Classes

Dynamics Model employ the equations of motion, which describe the relationships between the forces acting on the munition and the resulting motion [1]. In the model hierarchy TSONT has a dynamics model named Body Fixed Six DOF Dynamics Model, which is the model we used in Puma Trajectory Simulation. It implements equations of motion for a six degrees of freedom simulation, is classified under Six DOF Dynamic Models, which is a type of Rigid Body Dynamic Model.

2.2.4 Trajectory Simulation Functions

Functions that we use to compute a trajectory are also specified in TSONT. Trajectory Simulation Functions are seen in Figure 6. The functions which we have created the individuals of them to be used in Trajectory Simulation Federation are; 'Initialize Simulation', "Check Termination". The relation between functions and classes can be summarized as functions providing services to the classes.

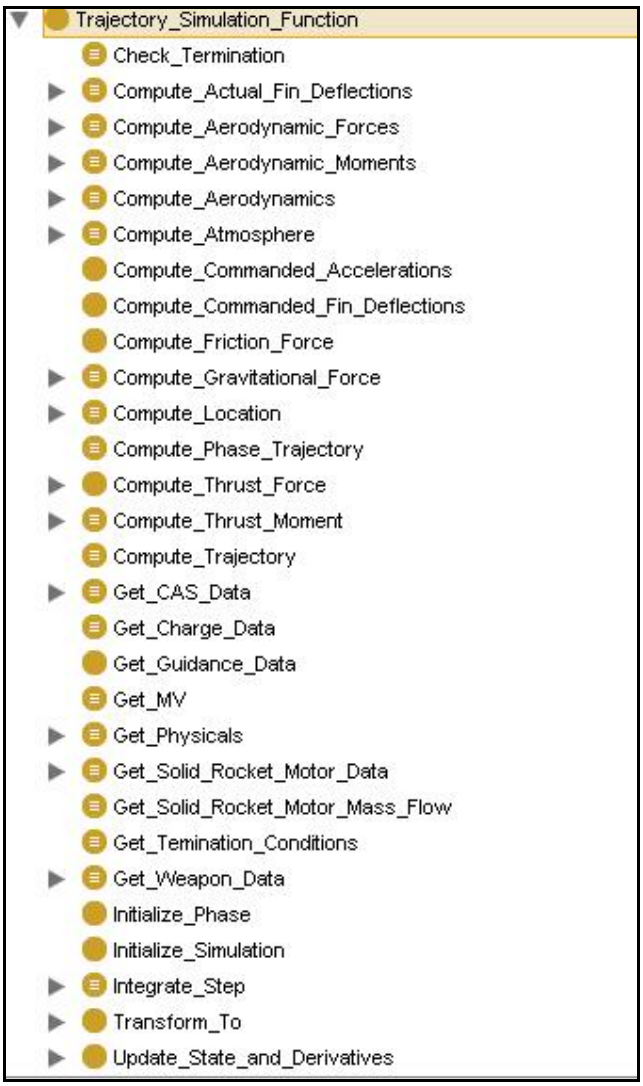


Figure 6 Trajectory Simulation Functions

2.2.5 Trajectory Simulation Quantities

Trajectory Simulation Quantities are assembled in two groups. In Body Fixed Six DOF Dynamics Model, which is used in Puma Trajectory Simulation, there are four types of Record Elements. These Record Elements are in the type of Vectoral Quantity. They are

'Translational Velocity in Body Coordinate System', 'Angular Rates in Body Coordinate System', 'Three Dimensional Position' and 'Euler Angles'.

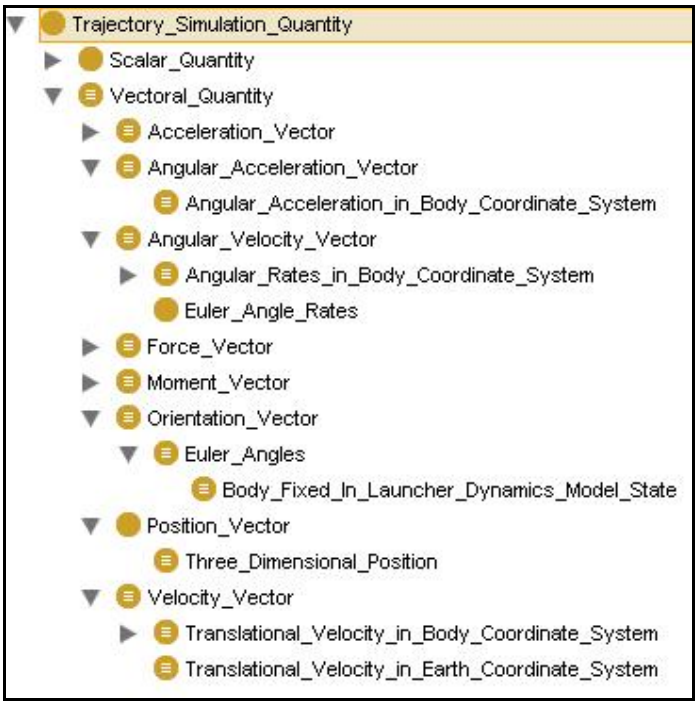


Figure 7 Trajectory Simulation Quantities

2.2.6 Trajectory Simulation Attributes

Trajectory Simulation Attribute defines a set of qualities of Trajectory Simulation Classes and Trajectory Simulation Objects such as the termination status of a trajectory or the ellipsoid of a location [11].

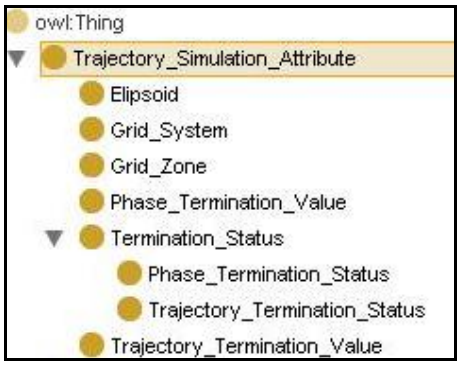


Figure 8 Trajectory Simulation Attributes

2.2.7 Trajectory Simulation Composite Data

Composite types are types whose values are composed or structured from simpler values [12]. They are used to group some data that forms a coherent construct. In developing trajectory simulation software, composite data types are widely used. TSONT tries to capture the composite data types that are used in the target reuse community. Trajectory Simulation Record and Trajectory Simulation Sequence are base Trajectory Simulation Composite Data types.

2.3 PUMA Trajectory Simulation

Puma Trajectory Simulation simulates the behavior and states of an air to ground guided bomb which has two flight phases, namely, unguided and guided. The bomb starts to flight in unguided phase and when safe separation is ensured phase changes to guided phase. In guided phase, the bomb flies to engage the target, so this phase is named guided. Development process of Puma Trajectory Simulation is visualized in Figure 9.

TSONT is transformed into UML by a model transformation tool OWL2UML [5], it is developed by Özdikiş. This transformation constitutes a platform independent framework architecture which includes class and sequence diagrams. Using this architecture, another architecture has been developed MATSIX – MATLAB 6DOF Trajectory Simulation Framework Architecture. Platform properties of MATLAB and its support for object oriented programming are reflected to the platform dependent framework architecture [25]. In Figure 9, this architecture is the output of "Infrastructure Specification" process. Then MATLAB 6 DOF Trajectory Simulation Framework has been developed, as "Infrastructure Implementation" part. Finally, by providing Puma-specific classes, Puma Trajectory Simulation is developed.

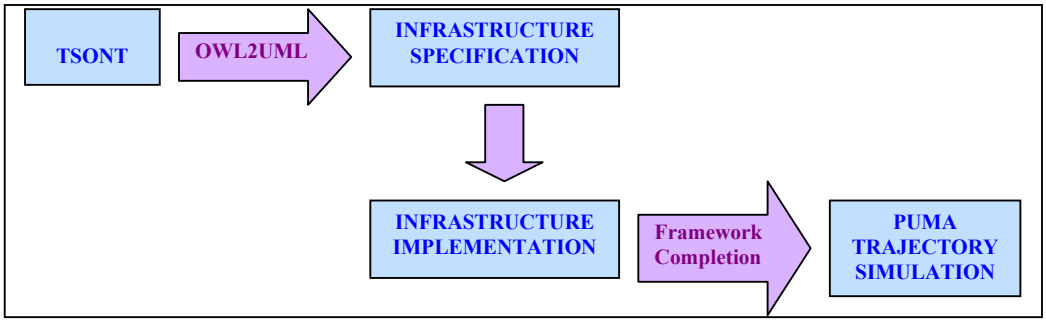


Figure 9 Puma Trajectory Simulation Development Process

2.4 High Level Architecture (HLA)

HLA is a component architecture which defines a component model. HLA combines simulations (federates) into a larger simulation (federation). Federates are components and federations are component based distributed applications.

HLA is architecture for distributed computer simulation systems. Different computer simulation systems can communicate via HLA at different platforms. This communication is managed and provided by Run Time Infrastructure (RTI).

In this thesis, HLA Standard that HLA Wrapper is based on is an international standard, IEEE 1516 HLA standard.

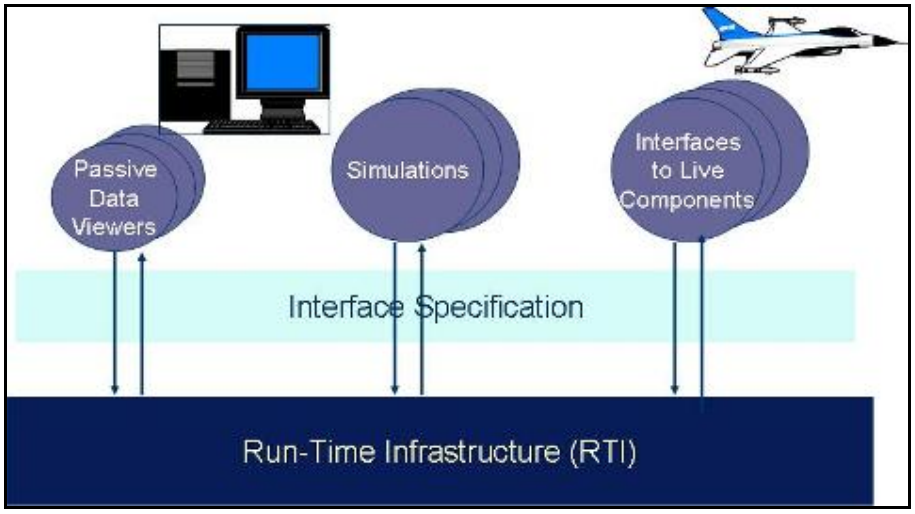


Figure 10 HLA Federation [http://www.cc.gatech.edu]

Object Model Template (OMT)

OMT is the Object Model Template, which provides a common framework for the communication between different HLA simulation applications. Federation Object Model (FOM) and Simulation Object Model (SOM) must comply with OMT.

Federation Object Model (FOM)

FOM is the HLA Federation Object Model, defines the model of the exchanged objects. An object is defined via its object class, attributes and its relation with the other objects in the federation.

Simulation Object Model (SOM)

SOM is the HLA Simulation Object Model. An object is defined via its object class, attributes and its relations. This shared object is used for a single federate.

Run-Time Infrastructure (RTI)

RTI is the Run-Time Infrastructure. It is the supporting middleware that manages the execution of the federation.

HLA Rules

These rules are extracted from IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules[15]. The HLA rules describe the responsibilities of federations and the federates that join.

- Federations shall have an HLA Federation Object Model (FOM), documented in accordance with the HLA Object Model Template (OMT).
- In a federation, all representation of objects in the FOM shall be in the federates, not in the run-time infrastructure (RTI).
- During a federation execution, all exchange of FOM data among federates shall occur via the RTI.
- During a federation execution, federates shall interact with the run-time infrastructure (RTI) in accordance with the HLA interface specification.
- During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time.
- Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA Object Model Template.
- Federates shall be able to update and/or reflect any attributes of objects in their SOM and send and/or receive SOM object interactions externally, as specified in their SOM.
- Federates shall be able to transfer and/or accept ownership of an attribute dynamically during a federation execution, as specified in their SOM.
- Federates shall be able to vary the conditions under which they provide updates of attributes of objects, as specified in their SOM.
- Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

Federation Execution Process (FEDEP)

Federation Development and Execution Process(FEDEP) IEEE 1516.3-2003, is a standardized and recommended process for developing interoperable HLA based federations. FEDEP defines the steps which will be followed while creating the federations.

FEDEP steps can be handled in four groups; initialization, declaration of objects and interests, information exchanging, terminating execution.

Initialization

- Create Federation Execution (Federation management)
- Join Federation Execution (Federation management)

Declaration of Objects and Interests

- Publish Object Class Attributes (Declaration management)
- Subscribe Object Class Attributes (Declaration management)

Information exchanging

- Update/Reflect Attribute Values (Object Management)
- Send/Receive Interaction (Object Management)
- Time Advance Request, Time Advance Grant (Time Management)
- Request Attribute Ownership Assumption (Ownership Management)
- Send Interaction with Regions (Data Distribution Management)

Terminating execution

- Resign Federation Execution (Federation Management)
- Destroy Federation Execution (Federation Management)

2.5 HLA Programming in Java

HLA may be programmed in many different programming languages. Regarding pRTI, these languages can be as following:

- C++
- Any computer language with an interface module written in C++
- Java
- Any computer language, wrapped in Java

Some parts of the information, which will be represented in this section are extracted from pRTI1516 Users Guide [24] and the code examples are derived from Puma and Exercise Manager federate code.

The FederateAmbassador and the RTIAmbassador

There are two main Java interfaces to deal with developing a federate, FederateAmbassador and RTIAmbassador. The FederateAmbassador interface is the interface through which the RTI communicates with the federate. The pRTI 1516 provides the convenience class FederateAmbassadorImpl which simply provides empty implementations of all the callback methods in the FederateAmbassador interface. When you develop your own federate you can

simply subclass FederateAmbassadorImpl and implement (override) the callbacks that you are interested in. RTIambassador is the class through which the federate communicates with RTI.

```
rti = RTI.getRTIambassador(hostName , portNumber);
```

hostName is the address of machine that Central RTI Component(CRC) is executed on, portNumber is the communicated port number on that machine. All communication between the federate and the RTI must go through the FederateAmbassador and the RTIambassador interfaces.

The Federation Object Model

The first thing that needs to be done is to create the federation execution. The Create Federation Execution service takes a FOM file as an argument.

```
try {
        final File fddFile;
        fddFile = new File("TrajectorySim.xml");
        rti.createFederationExecution(federationExecutionName,
        fddFile.toURI().toURL());
} catch (FederationExecutionAlreadyExists ignored) {
}//end of try/catch
```

The parameters are the name of the federation (federationExecutionName) and the URL representation of the FOM file to use (TrajectorySim.xml). When the federation is created the federate has to join the federation.

```
FederateHandle federateHandle;
String federateType = "ExerciseManagerFederate";
rti.joinFederationExecution(federateType,
federationExecutionName, this, null);
```

The first two parameters are the names of the federate and the federation to join, respectively. The third parameter is a reference to the FederateAmbassador to join. The last parameter is used to indicate which time representation you are using in your federation execution. In this case, we are using the default time representation.

Objects and interactions are used to exchange data between federates in the federation. Objects have attributes, and interactions have parameters, to describe their characteristics. Each object and attribute is represented by a handle.

```
private ObjectClassHandle UnguidedPhaseClassHandle ;
private AttributeHandle TransVelocityHandle;
private AttributeHandle AngularRatesHandle;
private AttributeHandle ThreeDPositionHandle;
private AttributeHandle EulerAnglesHandle;

UnguidedPhaseClassHandle =
rti.getObjectClassHandle("Puma_Unguided_Simulation_Phase");
```

```
TransVelocityHandle =
rti.getAttributeHandle(UnguidedPhaseClassHandle,
"Puma_Translational_Velocity_in_Body_Coordinate_System");

AngularRatesHandle =
rti.getAttributeHandle(UnguidedPhaseClassHandle,
"Puma_Angular_Rates_in_Body_Coordinate_System");

ThreeDPositionHandle =
rti.getAttributeHandle(UnguidedPhaseClassHandle,
"Puma_Three_Dimensional_Position");

EulerAnglesHandle =
rti.getAttributeHandle(UnguidedPhaseClassHandle,
"Puma_Euler_Angles");

private ObjectClassHandle GuidedPhaseClassHandle ;
private AttributeHandle PCanardDefHandle ;
private AttributeHandle PCanardDefRatesHandle ;

GuidedPhaseClassHandle =
rti.getObjectClassHandle("Puma_Guided_Simulation_Phase");

TransVelocityHandle =
rti.getAttributeHandle(GuidedPhaseClassHandle,
"Puma_Translational_Velocity_in_Body_Coordinate_System");

AngularRatesHandle =
rti.getAttributeHandle(GuidedPhaseClassHandle,
"Puma_Angular_Rates_in_Body_Coordinate_System");

ThreeDPositionHandle =
rti.getAttributeHandle(GuidedPhaseClassHandle,
"Puma_Three_Dimensional_Position");

EulerAnglesHandle =
rti.getAttributeHandle(GuidedPhaseClassHandle,
"Puma_Euler_Angles");

PCanardDefHandle =
rti.getAttributeHandle(GuidedPhaseClassHandle,
"Puma_Physical_Canard_Deflections");

PCanardDefRatesHandle =
rti.getAttributeHandle(GuidedPhaseClassHandle,
"Puma_Physical_Canard_Deflection_Rates");
```

The parameter of the getObjectClassHandle is the name of object class as specified in the FOM file. The first parameter in the getAttributeHandle is the object to which the parameter belongs and the second one is the name of the attribute as specified in the FOM file.

Each interaction and parameter is represented by a handle.

```
InteractionClassHandle interactioninitHandle;
ParameterHandle TransVelHandle ;
ParameterHandle ThreeDimHandle ;
ParameterHandle EulerAngHandle ;
ParameterHandle AngRateHandle ;
```

```
interactioninitHandle = rti.getInteractionClassHandle(
"Initialize_Puma_Simulation" );
TransVelHandle = rti.getParameterHandle( interactioninitHandle,
"Puma_Translational_Velocity_in_Body_Coordinate_System" );
ThreeDimHandle = rti.getParameterHandle( interactioninitHandle,
"Puma_Three_Dimensional_Position" );
EulerAngHandle = rti.getParameterHandle( interactioninitHandle,
"Puma_Euler_Angles" );
AngRateHandle = rti.getParameterHandle( interactioninitHandle,
"Puma_Angular_Rates_in_Body_Coordinate_System" );


InteractionClassHandle unguidedphasetermHandle;
ParameterHandle phasetermHandle ;
unguidedphasetermHandle = rti.getInteractionClassHandle(
"Puma_Check_UnguidedPhase_Termination" );
phasetermHandle = rti.getParameterHandle(
unguidedphasetermHandle, "Puma_Phase_Termination_Status" );

InteractionClassHandle guidedphasetermHandle;
ParameterHandle trajectorytermHandle ;
guidedphasetermHandle = rti.getInteractionClassHandle(
"Puma_Check_GuidedPhase_Termination" );
trajectorytermHandle = rti.getParameterHandle(
guidedphasetermHandle, "Puma_Trajectory_Termination_Status" );
```

The parameter of the getInteractionClassHandle call is the name of the interaction as specified in the FOM file. The first parameter in the getParameterHandle call is the interaction to which the parameter belongs and the second one is the name of the parameter.The handles are the federate's representation of interactions and parameters.These handles are used to send/receive interactions and to request/provide updates for objects.

Publishing Information and Subscribing to Information

The exchange of data is controlled by publishing of data and subscribing to data. For an interaction or for attributes of an object class; the sending federate must first publish it, which means that it tells everyone that it has some information and that it wants to share it. For a federate to receive interactions of a certain class or to receive attribute updates for an object class; it must subscribe to that interaction class or to attributes of that object class.

```
rti.subscribeInteractionClass( interactioninitHandle );
rti.subscribeObjectClassAttributes(UnguidedPhaseClassHandle,
unguidedAttributes);

rti.publishInteractionClass( interactioninitHandle );
rti.publishObjectClassAttributes(UnguidedPhaseClassHandle,
unguidedAttributes);
```

Other federates will now able to receive interactions or attribute updates sent by you , if they have already subscribed to.

Sending and Receiving Interactions

ParameterHandleValueMap should be created before sending interactions, and interaction's parameters and the values of them should be added to this map.

```
ParameterHandleValueMap parametermap =
rti.getParameterHandleValueMapFactory().create(15);

parametermap.put(TransVelHandle, (TVelocity.toString()).
getBytes());
parametermap.put(ThreeDimHandle, (TDPosition.toString()).
getBytes());
parametermap.put(AngRateHandle, (ARates.toString()).
getBytes());
parametermap.put(EulerAngHandle,(EAngles.toString()).
getBytes());
```

First parameter of put method is the handle of the interaction parameter and the second parameter is the up to date value of that parameter which you want to send within the interaction (must be converted to byte[]).

To send the interaction the following call is made:

```
rti.sendInteraction( interactioninitHandle, parametermap , null );
```

The first parameter is the interaction class handle, the second is the ParameterHandleValueMap, holding the parameter values of the interaction and the third is a user-supplied tag, in this case set to null.

ReceiveInteraction is called by the RTI when an interaction sent by another federate is to be delivered to your federate.

```
public void receiveInteraction(InteractionClassHandle
interactionClass, ParameterHandleValueMap parametermap, byte[]
tag, OrderType arg3, TransportationType arg4)
```

The first parameter is the class handle of the received interaction and the second one is the interaction's parameters. To get the parameter values out of the ParameterHandleValueMap you can for example go through the map using a for loop like this:

```
String newMember = new String((byte[])parametermap.
get(TransVelHandle));
StringTokenizer str = new StringTokenizer(newMember , ",[] ");
    while (str.hasMoreElements()) {
          String token = str.nextToken();
          Double d = Double.parseDouble(token);
          TVelocity.addElement(d.intValue());
    }
```

Updating Attribute Values

AttributeHandleSet should be created first, then attribute handles are added to that attribute handle set as seen in the following.

```
AttributeHandleSetFactory guidedFactory = rti.
getAttributeHandleSetFactory();
AttributeHandleSet guidedAttributes = guidedFactory.create();

guidedAttributes.add(TransVelocityHandle);
guidedAttributes.add(AngularRatesHandle);
guidedAttributes.add(ThreeDPositionHandle);
guidedAttributes.add(EulerAnglesHandle);
guidedAttributes.add(PCanardDefHandle);
guidedAttributes.add(PCanardDefRatesHandle);
```

Then federate requests update for the attributes of the object class that is the first parameter of requestAttributeValueUpdate. Second parameter is the attribute handle set, it handles just the attributes added to this handle set. Federate requests update for the attributes handles of which are added to the attribute handle set. By this way, federate may request update for some of the attributes or for all of them.

```
rti.requestAttributeValueUpdate(GuidedPhaseClassHandle,
guidedAttributes, null);
```

AttributeHandleValueMap should be created before sending updates for attribute values. Attribute handles and the attribute values are added to this map.

```
AttributeHandleValueMap _attributeValues;
_attributeValues = rti.getAttributeHandleValueMapFactory().
create(1);
_attributeValues.put(TransVelocityHandle, TVelocity.toString().
getBytes());
_attributeValues.put(AngularRatesHandle, ARates.toString().
getBytes());
_attributeValues.put(ThreeDPositionHandle, TDPosition.toString().
getBytes());
_attributeValues.put(EulerAnglesHandle, EAngles.toString().
getBytes());
_attributeValues.put(PCanardDefHandle, PCanardDef.toString().
getBytes());
_attributeValues.put(PCanardDefRatesHandle, PCanardDefRates.
toString().getBytes());

rti.updateAttributeValues(guidedInstance, _attributeValues, null);
```

When requestAttibuteValueUpdate method of RTI is called at the federate, provideAttributeValueUpdate method of the other federate is called. ProvideAttributeValueUpdate method updates the attribute values and federate calls updateAttributeValues method of RTI. With this call, RTI calls reflectAttributeValues method of the federate which has sent the update request. In this manner, process of updating attribute values finishes.

CHAPTER 3

TSONT TO FEDERATION OBJECT MODEL

This chapter presents TSONT from the perspective of an HLA developer. First, we look into TSONT in detail to discover the objects, their attributes and relations which we will simulate in the application. We investigate TSONT using an ontology editor named Protégé [26]. Then we deal with the individuals in Protégé. Individuals are created depending upon the Puma Trajectory Simulation requirements. Individuals are the exchanged objects and interactions of the federation from the simulation application point of view. When we simulate the ontology in the HLA application, we look into the ontology down to a certain depth. This depth is defined by the object attributes. Finally, we have the FOM, which defines the object classes, their attributes, and interactions which are exchanged by the federates in the federation. The attributes we put into the FOM are updated and exchanged in the federate code.

3.1 Analyzing TSONT

TSONT has a hierarchy which is composed of the high level entities Attribute, Class, Composite Data, Function, Object and Quantity as described in Section 2.2 . While developing and analyzing TSONT a tool called Protégé is used.

Protégé is a tool developed by Stanford University. It provides a graphical environment to facilitate the communication with the domain experts besides enabling an integrated formalization of the captured conceptualization while constructing graphical representation of ontology [14].

Aim of analyzing TSONT is to be familiarized with the functions, objects and the relations for being able to create the individuals according to the requirements of Puma Federate. Trajectory Simulation Class has a view as shown in Figure 11.

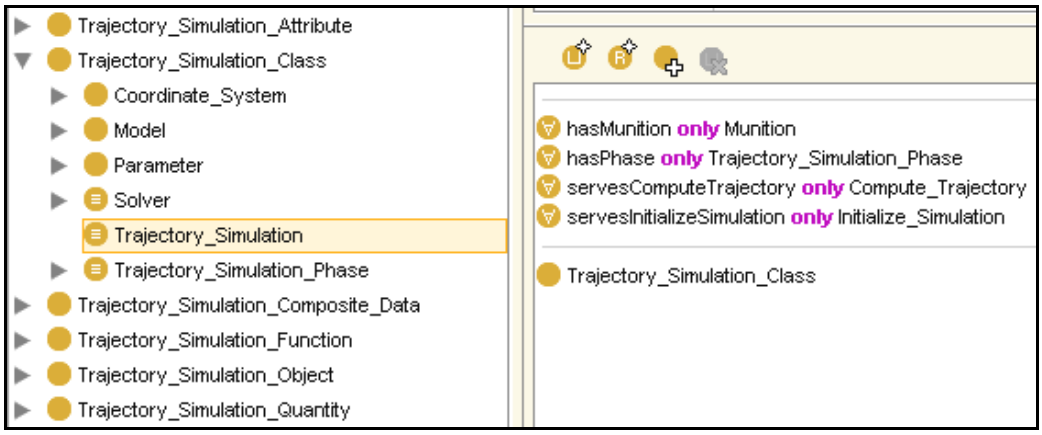


Figure 11 Trajectory Simulation Class

Trajectory Simulation Class is composed of an object of type Munition, a phase of type Trajectory Simulation Phase, a function of type Compute Trajectory and a function of type Initialize Simulation.

In our simulation the subtype of the Munition is Bomb and the subtype of the Bomb is Guided Bomb. Guided bombs have a two-phased flight profile. The first phase is the unguided flight that is for safe separation. The second phase is the guided flight directed to the target. Guided Bomb is seen in Figure 12. The phases are defined in detail in the next section.

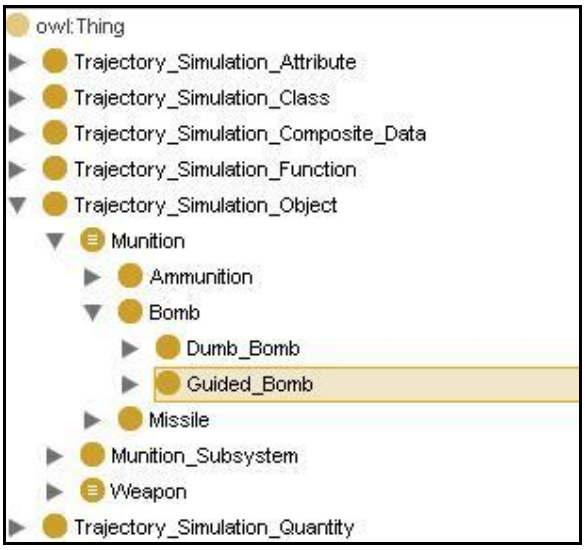


Figure 12 Guided Bomb

Dynamics models in trajectory simulations are classified either as point mass dynamics models or as rigid body dynamics models. A dynamics model that implements equations of motion for a six degrees of freedom simulation, namely Body Fixed Six DOF Dynamics Model, is classified under Six DOF Dynamic Models, which is a type of Rigid Body

Dynamics Model. Body Fixed Six DOF Dynamics Model State that is seen in Figure 13 is the model state which is exchanged among federates in the Unguided phase of the flight.

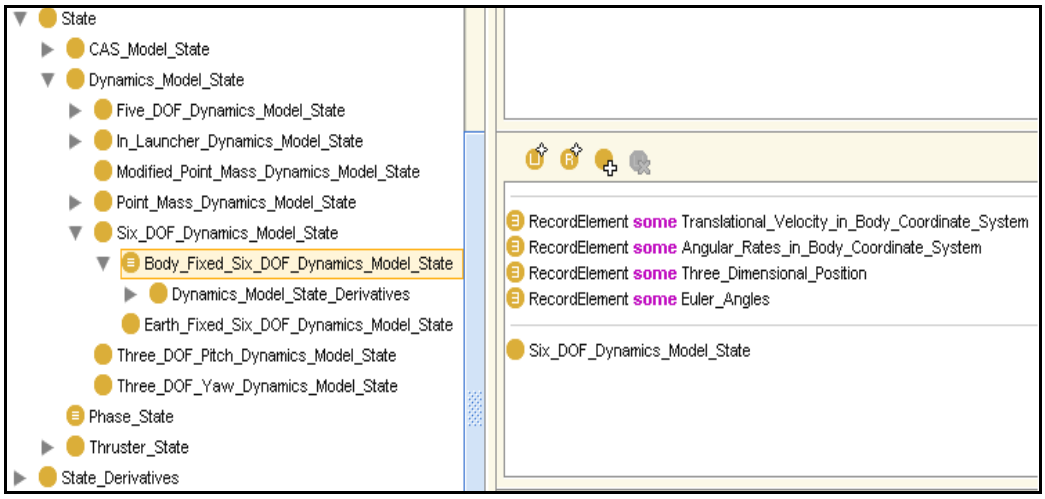


Figure 13 Body Fixed Six DOF Dynamics Model State

Body Fixed Six DOF Dynamics Model State has four Record Elements as listed below:

- Translational Velocity in Body Coordinate System is a vector with 3 records which are in the type of Velocity.
- Angular Rates in Body Coordinate System is a vector with 3 records which are in the type of Angular Velocity.
- Three Dimensional Position is a vector with 3 records which are in the type of Length.
- Euler Angles is a vector with 3 records which are in the type of Plane Angle.

Second Order CAS Model State that is seen in Figure 14 is the model state which is exchanged among federates in the Guided phase of the flight.

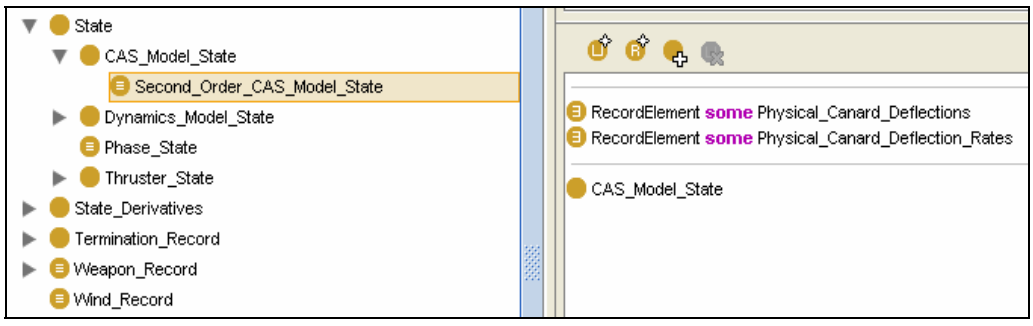


Figure 14 Second Order CAS Model State

3.2 Creating Individuals

Creating individuals is the part of TSONT specific to the Puma Federate. In the previous sections we overviewed TSONT. In this section, creation of the individuals is outlined by taking the Puma Federate requirements into consideration.

First, Puma Simulation is introduced and we go deeply into the other levels. As we said before there is a class Trajectory Simulation in type of Trajectory Simulation Class. While starting the creation of individuals, the first individual we create is Puma Simulation, which is a type of Trajectory Simulation as seen in Figure 15.

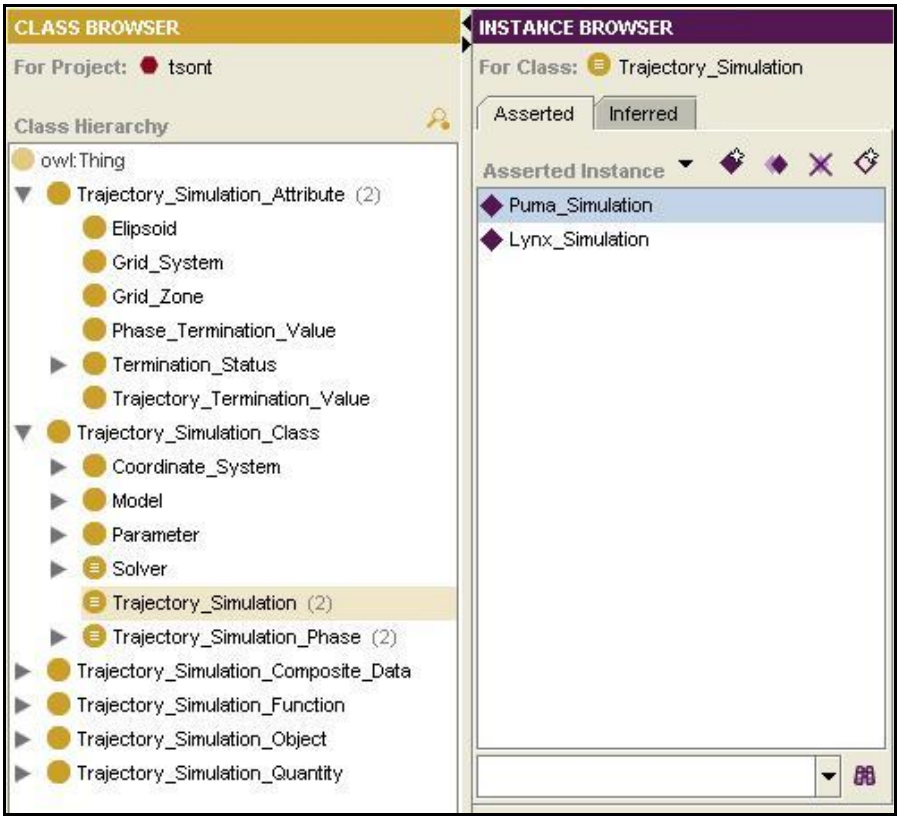


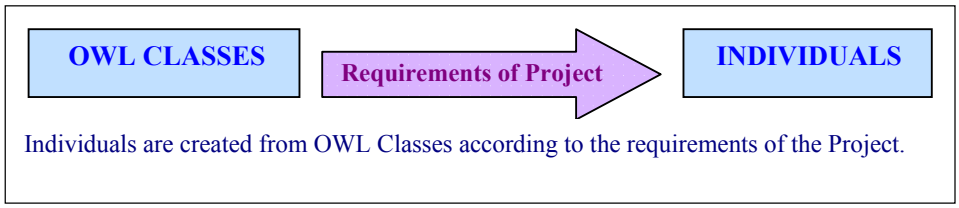
Figure 15 Puma Simulation Individual

Then the sub entities of Puma Simulation are created such as its objects, functions and phases. All individuals will be listed in detail in the next section.

3.3 TSONT Classes through TSONT Individuals

We have defined TSONT Classes in Section 3.1 and TSONT Individuals in Section 3.2. In this section we will detail the relation between OWL Classes and Individuals.

An OWL ontology includes descriptions of classes, properties and their instances [26]. According to this definition, classes in the ontology are called as OWL Classes and the instances are called as the Individuals. OWL Classes have been defined in the phase of ontology(TSONT) creation. Individuals are created in the present work because of building and running simulations. OWL Classes includes a variety of attributes, classes, composite data, functions, objects and quantities as mentioned in the Section 2.2 TSONT.



Individuals are generated using OWL Classes specifically for a work. For understanding the relation between OWL Classes and Individuals clearly, it will be beneficial to take a look at this thesis work. An air to ground bomb simulation is represented here. There is a munition with initial values. These values change by time till the munition reaches the target. Munition starts a free flight, named Unguided Phase and then the phase switches Guided Phase. Unguided Phase has Dynamics_Model, and Guided Phase has Dynamics_Model and CAS_Model. Munition has two interactions named Initialize_Simulation and Check_Termination which are in the type of Trajectory_Simulation_Function. Necessary OWL Classes for the present thesis and TSONT Classes – Individuals relation can also be seen in Table 1 .

Table 1 TSONT Classes – Individuals Traceability Matrix

OWL Classes	Individuals
Trajectory_Simulation	Puma_Simulation
GPS_Guided_Bomb	Puma
Trajectory_Simulation_Phase	Puma_Unguided_Simulation_Phase
Guided_Phase	Puma_Guided_Simulation_Phase
Body_Fixed_Six_DOF_Dynamics_Model	Puma_Dynamics_Model
Body_Fixed_Six_DOF_Dynamics_Model_ State	Puma_DM_State
Angular_Rates_in_Body_Coordinate_ System	Puma_Angular_Rates_in_Body_Coordinate_ System
Euler_Angles	Puma_Euler_Angles
Three_Dimensional_Position	Puma_Three_Dimensional_Position
Translational_Velocity_in_Body_ Coordinate_System	Puma_Translational_Velocity_in_Body_ Coordinate_System
Second_Order_CAS_Model	Puma_CAS_Model
CAS_Model_State	Puma_Second_Order_CAS_Model_State
Physical_Canard_Deflections	Puma_ Physical_Canard_Deflections
Physical_Canard_Deflection_Rates	Puma_ Physical_Canard_Deflection_Rates
Initialize_Simulation	Initialize_Puma_Simulation
Phase_State	Puma_Unguided_Phase_State
Check_Termination	Puma_Check_UnguidedPhase_Termination
Check_Termination	Puma_Check_GuidedPhase_Termination
Phase_Termination_Status	Puma_Phase_Termination_Status
Trajectory_Termination_Status	Puma_Trajectory_Termination_Status

3.4 Developing the FOM

As we introduced in Section 2.4 FOM part, FOM is the HLA Federation Object Model, and it defines the models of the exchanged objects. In this study, we have created an individual called Puma Simulation which is an instance of the OWL Class Trajectory Simulation. Counterpart of Puma Simulation in the application development process, is thought of as the Trajectory Simulation Federation. This federation includes two federates, Puma Federate and Exercise Manager Federate. Objects, attributes, interactions and interaction parameters which are exchanged by Puma Federate and Exercise Manager Federate are defined in FOM. In Table 2, Individuals created in TSONT and their counterparts in FOM are summarized.

Table 2 Individuals – FOM Traceability Matrix

Individuals	**FOM Counterparts**
Puma_Unguided_Simulation_Phase	Puma_Unguided_Simulation_Phase ObjectClass
Puma_Guided_Simulation_Phase	Puma_Guided_Simulation_Phase ObjectClass
Puma_Angular_Rates_in_Body_ Coordinate_System	Attribute of Puma_Unguided_Simulation_Phase Class and Puma_Guided_Simulation_Phase Class
Puma_Euler_Angles	Attribute of Puma_Unguided_Simulation_Phase Class and Puma_Guided_Simulation_Phase Class
Puma_Three_Dimensional_Position	Attribute of Puma_Unguided_Simulation_Phase Class and Puma_Guided_Simulation_Phase Class
Puma_Translational_Velocity_in_ Body_Coordinate_System	Attribute of Puma_Unguided_Simulation_Phase Class and Puma_Guided_Simulation_Phase Class
Puma_ Physical_Canard_Deflections	Attribute of Puma_Guided_Simulation_Phase Class
Puma_ Physical_Canard_Deflection_ Rates	Attribute of Puma_Guided_Simulation_Phase Class
Initialize_Puma_Simulation	Initialize_Puma_Simulation InteractionClass
Puma_Unguided_Phase_State	Parameter of Initialize_Puma_Simulation Class
Puma_Check_UnguidedPhase_ Termination	Puma_Check_UnguidedPhase_Termination InteractionClass
Puma_Phase_Termination_Status	Parameter of Puma_Check_UnguidedPhase_Termination Class
Puma_Check_GuidedPhase_ Termination	Puma_Check_GuidedPhase_Termination InteractionClass
Puma_Trajectory_Termination_Status	Parameter of Puma_Check_GuidedPhase_Termination Class

There are two ways of developing FOM, one is manually, and the other is by using the OWL2OMT tool.

3.4.1 Developing the FOM Manually

There has to be two object classes in the FOM which are derived from the phases of the Puma Simulation individual. These object classes are Unguided and Guided Simulation Phase object classes. When Puma Federate is launched it starts a free flight, named Unguided Phase of the Puma Federate. This phase is simulated with an object class named Puma_Unguided_ Simulation Phase which is derived from an individual with the same name; the attributes of this object class are obtained from Puma_DM_State individual. Attributes of Puma_Unguided_Simulation_Phase can be seen in FOM Counterpart column of Table 2.

To develop the FOM manually, first, Puma_Unguided_Simulation_Phase is written in FOM between "objectClass" tags and the attributes Puma_Angular_Rates_in_Body_ Coordinate_System, Puma_Euler_Angles, Puma_Three_Dimensional_Position, and Puma_Translational_Velocity_in_Body_Coordinate_System are written under "attribute" tag.

```
<objectClass name="Puma_Unguided_Simulation_Phase"
sharing="Neither">
<attribute name="Puma_Angular_Rates_in_Body_Coordinate_System"
dataType="HLAvariableArray" dimensions="NA"
transportation="HLAreliable" order="Receive"/>
<attribute name="Puma_Three_Dimensional_Position"
dataType="HLAvariableArray" dimensions="NA"
transportation="HLAreliable" order="Receive"/>
<attribute name="Puma_Euler_Angles" dataType="HLAvariableArray"
dimensions="NA" transportation="HLAreliable" order="Receive"/>
<attribute name="Puma_Translational_Velocity_in_Body_Coordinate_
System" dataType="HLAvariableArray" dimensions="NA"
transportation="HLAreliable" order="Receive"/>
</objectClass>
```

Figure 16 Puma_Unguided_Simulation_Phase Object Class in FOM

Second, object class simulates the Guided Phase of the Puma Federate; in this phase object class is Puma_Guided_Simulation_Phase and has the attributes which are also in Puma_Unguided_Simulation_Phase. In addition to these attributes, Puma_Guided_ Simulation_Phase has two more attributes, Puma_Physical_Canard_Deflections and Puma_Physical_Canard_Deflection_Rates, which are RecordElements of Puma_Second_Order_CAS_Model_State individual. Object class name is written between "objectClass" tags and attributes are written under "attribute" tag.

```
<objectClass name="Puma_Guided_Simulation_Phase" sharing="Neither">
<attribute name="Puma_Angular_Rates_in_Body_Coordinate_System"
dataType="HLAvariableArray" dimensions="NA"
transportation="HLAreliable" order="Receive"/>
<attribute name="Puma_Three_Dimensional_Position"
dataType="HLAvariableArray" dimensions="NA"
transportation="HLAreliable" order="Receive"/>
<attribute name="Puma_Euler_Angles" dataType="HLAvariableArray"
dimensions="NA" transportation="HLAreliable" order="Receive"/>
<attribute name="Puma_Translational_Velocity_in_Body_Coordinate_
System" dataType="HLAvariableArray" dimensions="NA"
transportation="HLAreliable" order="Receive"/>
<attribute name="Puma_Physical_Canard_Deflections"
dataType="HLAvariableArray" dimensions="NA"
transportation="HLAreliable" order="Receive"/>
<attribute name="Puma_Physical_Canard_Deflection_Rates"
dataType="HLAvariableArray" dimensions="NA"
transportation="HLAreliable" order="Receive"/>
</objectClass>
```

Figure 17 Puma_Guided_Simulation_Phase Object Class in FOM

Besides Puma Federate, in the Trajectory Simulation federation there is another federate named Exercise Manager. This federate initializes the flight simulation by sending an interaction named Initialize_Puma_Simulation. Parameters of this interaction are RecordElements of Puma_Unguided_Phase_State. These parameters are received by Puma Federate and are used to calculate the trajectory from launch to impact. Interaction name is written between “interactionClass” tags and parameters are written under “parameter” tag.

```
<interactionClass name="Initialize_Puma_Simulation"
sharing="PublishSubscribe" dimensions="NA"
transportation="HLAreliable" order="TimeStamp" semantics="NA">
<parameter name="Puma_Translational_Velocity_in_Body_Coordinate_
System" datatype="HLAvariableArray" semantics="NA"/>
<parameter name="Puma_Three_Dimensional_Position"
datatype="HLAvariableArray" semantics="NA"/>
<parameter name="Puma_Euler_Angles" datatype="HLAvariableArray"
semantics="NA"/>
<parameter name="Puma_Angular_Rates_in_Body_Coordinate_System"
datatype="HLAvariableArray" semantics="NA"/>
</interactionClass>
```

Figure 18 Initialize_Puma_Simulation Interaction Class in FOM

Puma_Check_UnguidedPhase_Termination interaction is an interaction that is sent from Puma Federate to Exercise Manager Federate. It has a parameter named “Puma_Phase_Termination_Status” which is derived from the individual with the same name. Puma Federate sends this interaction when it will destroy the Puma_Unguided_Simulation_Phase object. When Unguided Phase finishes and federate passes through the Guided Phase, it publishes Puma_Guided_Simulation_Phase object and Exercise Manager Federate subscribes to Puma_Guided_Simulation_Phase object.

```
<interactionClass name="Puma_Check_UnguidedPhase_Termination"
sharing="PublishSubscribe" dimensions="NA"
transportation="HLAreliable" order="TimeStamp" semantics="NA">
<parameter name="Puma_Phase_Termination_Status"
datatype="HLAboolean" semantics="NA"/>
</interactionClass>
```

Figure 19 Puma_Check_UnguidedPhase_Termination Interaction Class in FOM

Then Puma Federate registers this object instance to the federation. When the object is registered, it is discovered at Exercise Manager Federate side. So this interaction provides to track the trajectory information of Puma_Guided_Simulation_Phase object and besides it triggers to give up tracking the trajectory information of Puma_Unguided_Simulation_Phase object.

Last interaction is Puma_Check_GuidedPhase_Termination, when this interaction is sent from Puma Federate to Exercise Manager Federate, it means that the munition has reached the target, and this is the end of the trajectory. Parameter of this interaction is Puma_Trajectory_Termination_Status.

```
<interactionClass name="Puma_Check_GuidedPhase_Termination"
sharing="PublishSubscribe" dimensions="NA"
transportation="HLAreliable" order="TimeStamp" semantics="NA">
<parameter name="Puma_Trajectory_Termination_Status"
datatype="HLAboolean" semantics="NA"/>
</interactionClass>
```

Figure 20 Puma_Check_GuidedPhase_Termination Interaction Class in FOM

Puma Trajectory Simulation is linked to RTI's API by the HLA wrapper conforming to this federation object model.

As we have said in the upper section, there are two ways of developing FOM. First of all, object classes and interactions which will be exchanged between federates should be defined. According to this definition, as depicted in this section, FOM can be generated manually by writing these object classes, their attributes, interaction classes and their parameters into an Extensible Markup Language (XML) file. The other way is using OWL2OMT tool, but in this method we need to know the definition mentioned above, for being able to use the tool. We'll generate FOM, in conformity with the rules in the definition.

3.4.2 Developing the FOM with OWL2OMT

Generating FOM has been semi-automated by using OWL2OMT tool. Definition of the individuals is the only necessity, to be able to use the tool.

General Overview of OWL2OMT

OWL2OMT tool has a main application window named "OWL2OMT". Buttons on this frame are in the following order;

1. Load Configuration
2. Save Configuration
3. Add OWLClass Tab
4. Add OWLDatatypeProperty Tab
5. Add OWLObjectProperty Tab
6. Add OWLIndividual Tab
7. Remove Tab
8. Run Transformation

To generate a FOM from individuals, steps below should be followed:

1. Click "Load Configuration", select a configuration file and click open.

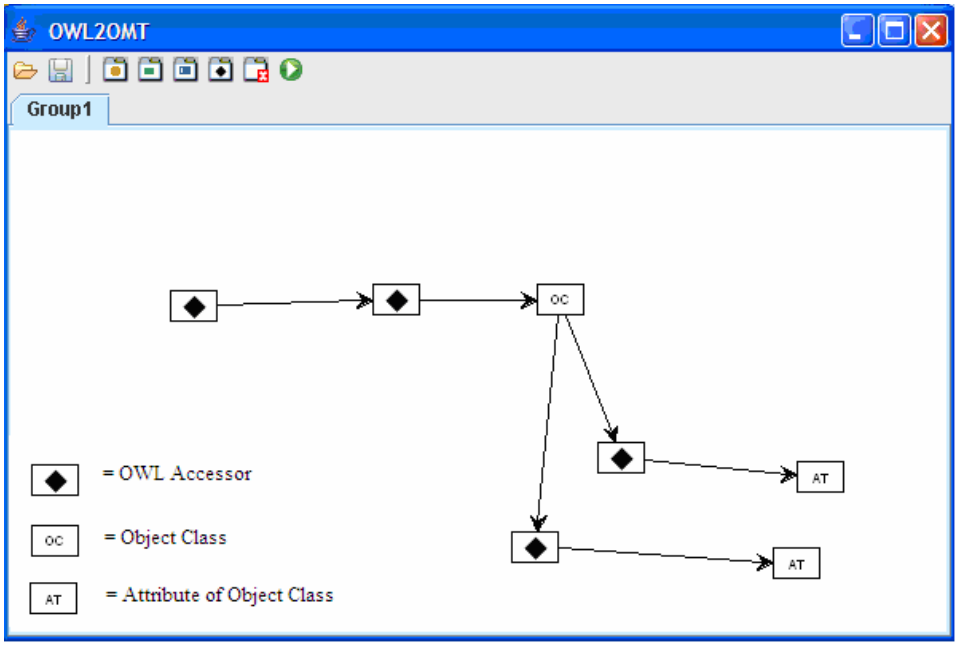


Figure 21 Configuration Loaded State

2. Except the buttons mentioned above, there is a chance to change configuration or other settings, as seen in Figure 22. Right clicking on an icon in the configuration area is effectual for this action.

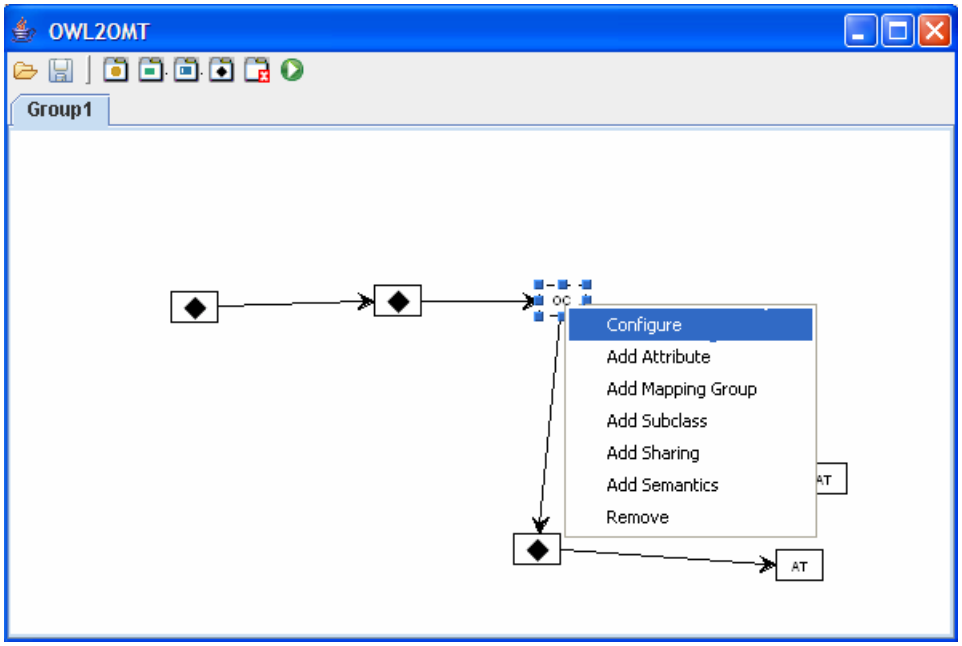


Figure 22 Changing a Setting in Configuration

3. If the configuration is sufficient for transformation, click “Run Transformation” button. Transformation begins after selecting a “Source OWL file”. When it finishes, an information message appears, “Transformation is done”. The application asks for a file name to save the result as a FOM document, and displays the information message “OMT file saved”. Produced file includes the same format and content with the one which we produce manually.

Generating FOM for Trajectory Simulation Federation

In the previous part, usage of OWL2OMT is summarized. In this part of the section, steps of generating a FOM particular to Trajectory Simulation Federation will be expressed.

1. On main window, click "Add OWLIndividual Tab". First OWL Acessor is created. Right click and select "Configure", configure OWL filter by setting the individual name to Puma_Simulation. Right click on the first OWL Accessor, select "Add Mapping Group". In this manner second OWL Accessor is created.

2. Right click on the second OWL Accessor, configure the Accessor by setting the property name to "hasPhase".

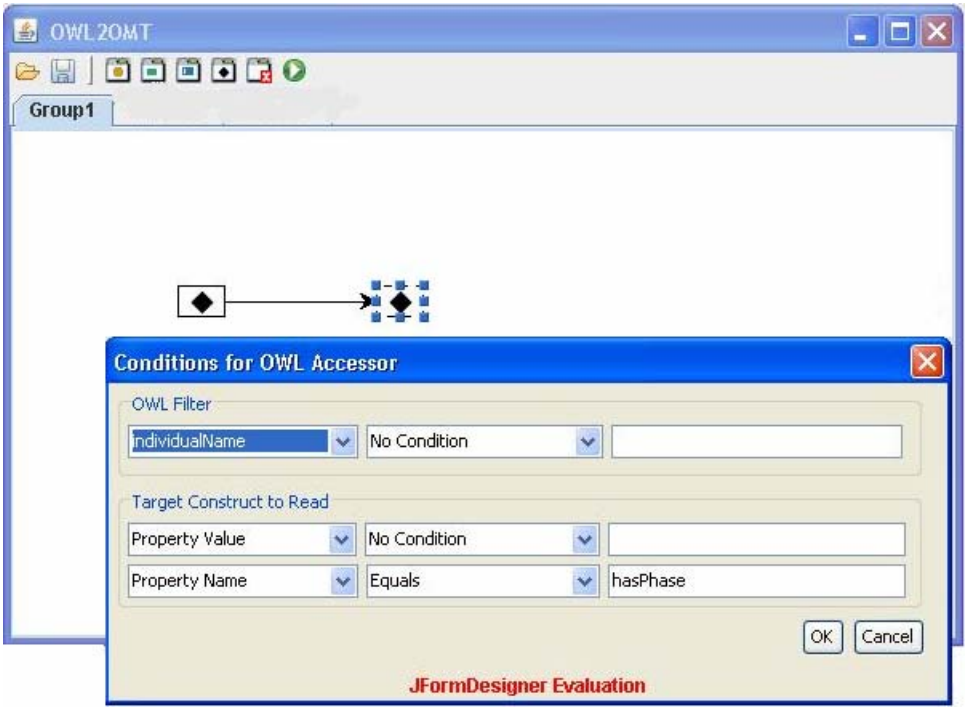


Figure 23 Generating FOM - hasPhase

Right click on the second OWL Accessor, and click "Add Object Class". With this action, Puma_Unguided _Simulation_Phase and Puma_Guided _Simulation_Phase will be added to FOM as object classes.

3. Right click on the ObjectClass, select "Add Mapping Group", right click new OWL Accessor, configure property name by setting it to "hasDynamicsModel".

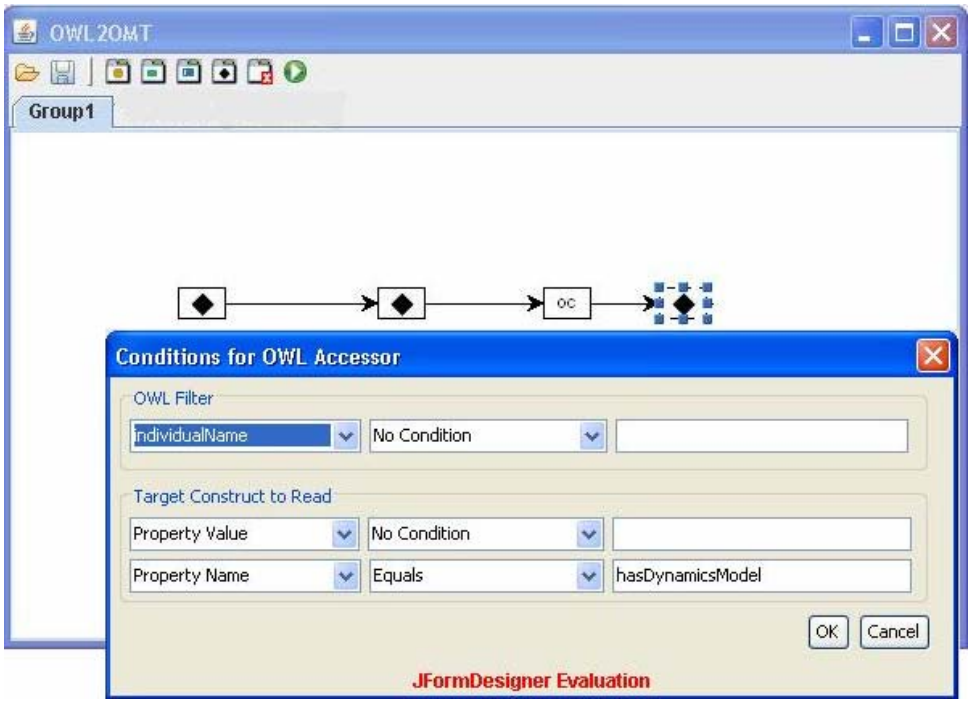


Figure 24 Generating FOM – hasDynamicsModel

Right click, select "Add Mapping Group", configure the lastly created OWL Accessor by setting the property name to "hasState". Right click this OWL Accessor, select "Add Attribute". Right click Attribute, configure as property name equals "RecordElement". This step provided to set RecordElements of Puma_DM_State individual as attributes of Puma_ Unguided_Simulation_Phase and Puma_Guided_Simulation_Phase object classes.

4. Right click on the ObjectClass, select "Add Mapping Group", right click new OWL Accessor, configure it by setting property name to "hasCASModel". Right click, select "Add Mapping Group", configure the last created OWL Accessor by setting the property name to "hasState". Right click on this OWL Accessor, select "Add Attribute".

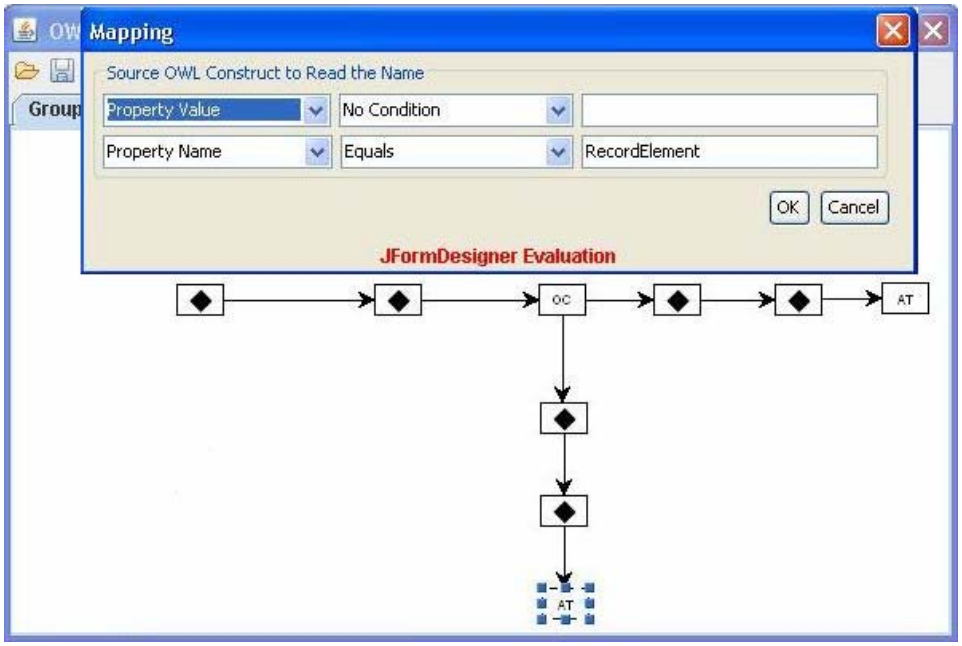


Figure 25 Generating FOM - Setting Attributes of Object Class

Right click Attribute, configure as property name equals "RecordElement". This step provided to set RecordElements of Puma_Second_Order_CAS_Model_ State individual as attributes of Puma_Guided_Simulation_Phase object class.

5. Right click on the first OWL Accessor which represents Puma_Simulation individual. Select "Add Mapping Group", configure the last created OWL Accessor by setting property name to "servesInitializeSimulation". Then right click and select "Add Interaction Class". Accordingly, an interaction class Initialize_Puma_ Simulation (an instance of servesInitializeSimulation) will be added to FOM with this configuration.

6. Right click on Interaction Class, select "Add Mapping Group", configure the last created OWL Accessor by setting property name to "inPhaseState". Right click on the OWL Accessor and select "Add Parameter". Configure the Parameter by setting its property name to RecordElement. Thus, RecordElements of Puma_Unguided_Phase_State are added to the FOM as parameters of the interaction class Initialize_Puma_ Simulation.

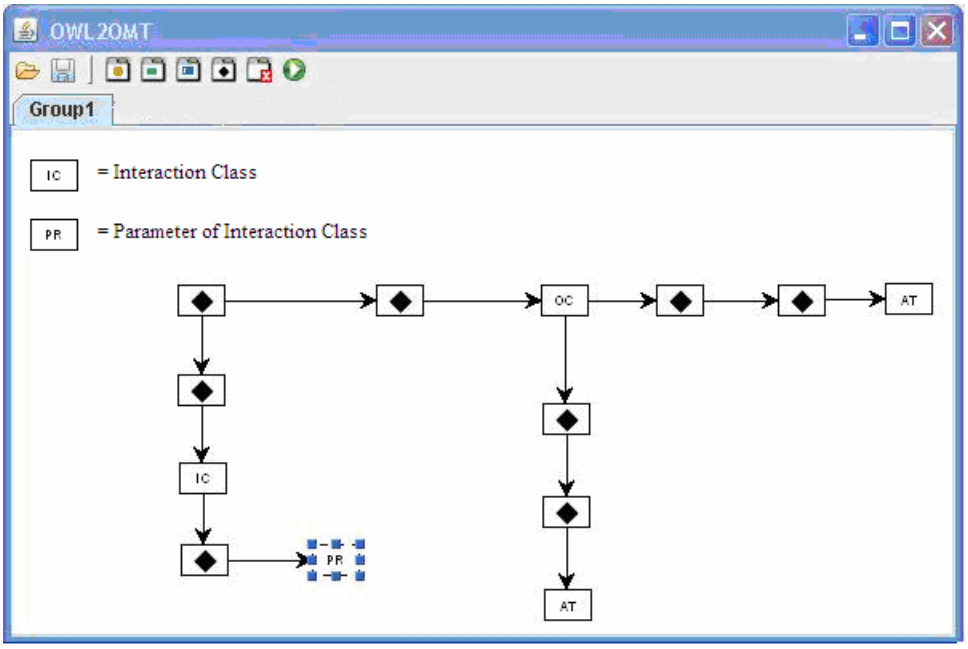


Figure 26 Generating FOM - Setting Parameters of Interaction Class

7. On main window, click "Add OWLIndividual Tab". A new group frame is opened and first OWL Accessor is created. Right click on the first OWL Accessor, select "Configure", configure OWL filter by setting individual name to Puma_Check_UnguidedPhase_Termination.

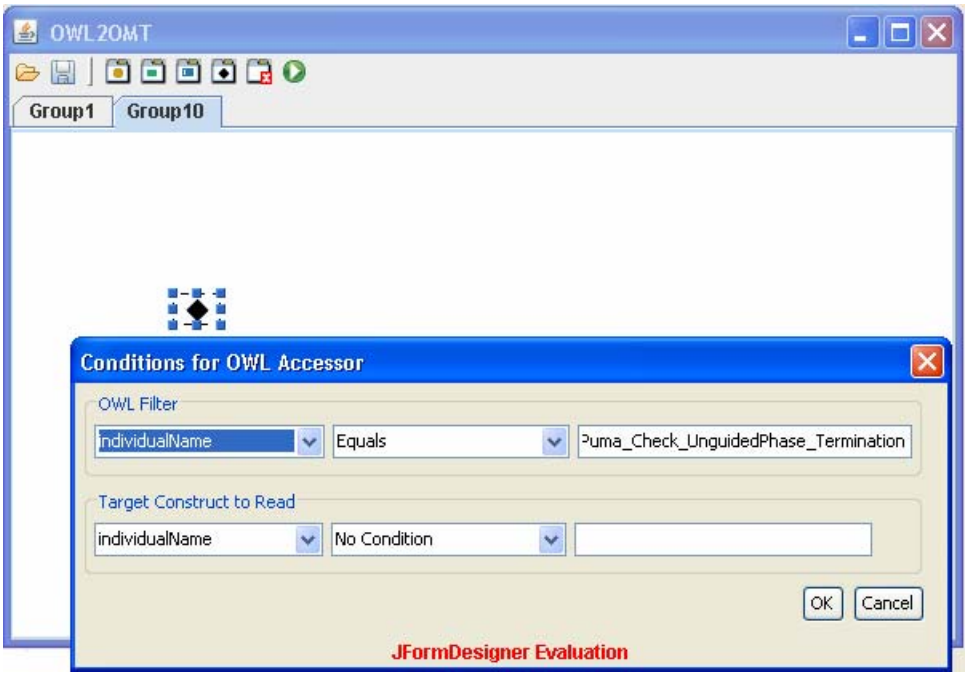


Figure 27 Generating FOM – Puma_Check_UnguidedPhase_Termination

8. Right click on the first OWL Accessor, select “Add Interaction Class”. According to this action, an interaction class Puma_Check_UnguidedPhase_Termination will be added to FOM. Right click on the Interaction Class, select “Add Parameter”. Right click on the Parameter, configure it by setting the property name to “outTerminationStatus”. Instance of outTerminationStatus is Puma_Phase_ Termination_Status, so this individual is set as parameter of this interaction class.

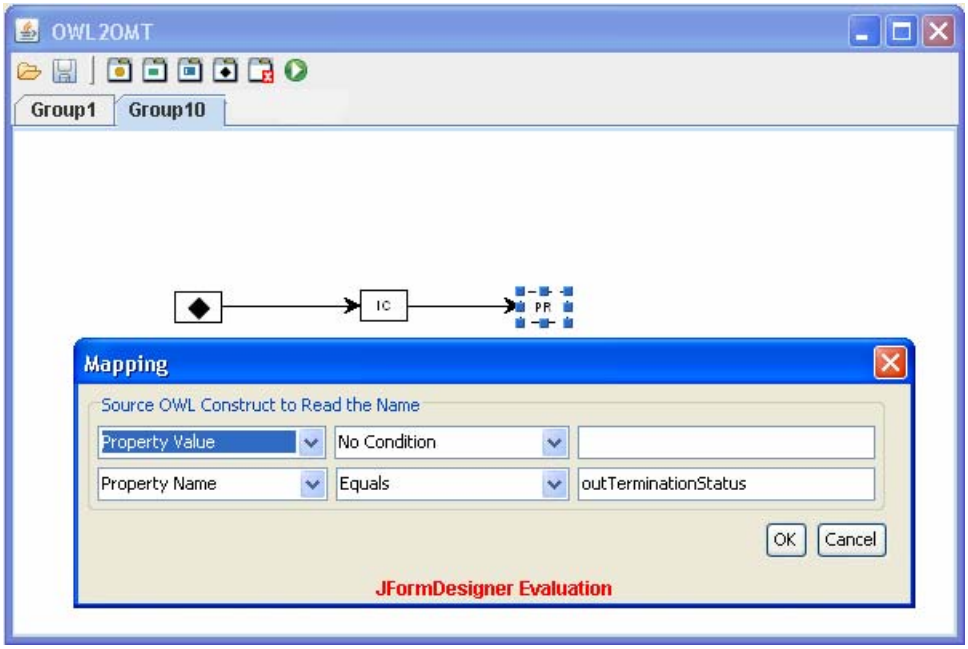


Figure 28 Generating FOM – Termination Status from Unguided Phase

9. On main window, click “Add OWLIndividual Tab”. A new group frame is opened and first OWL Accessor is created. Right click onthe first OWL Accessor, select “Configure”, configure OWL filter by setting individual name to Puma_Check_GuidedPhase_Termination in FOM.

10. Right click on the first OWL Accessor, select “Add Interaction Class”. As a result of this action, an interaction class Puma_Check_GuidedPhase_Termination will be added to FOM. Right click on the Interaction Class, select “Add Parameter”. Right click on the Parameter, configure its property name by setting to “outTerminationStatus”. Instance of outTerminationStatus is Puma_Trajectory_ Termination_Status, so this individual is set as parameter of this interaction class in FOM.

11. Steps thus far, explained preparing the configuration file which is necessary for transformation. Save the configuration file and then click “Run Transformation” button.

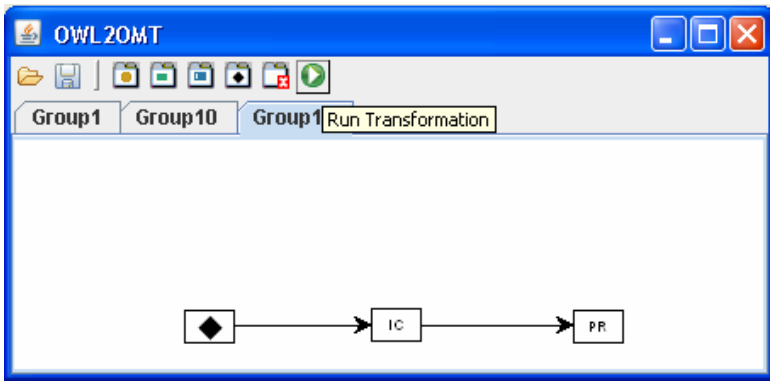


Figure 29 Generating FOM – Run Transformation

12. OWL2OMT asks for a “Source OWL file”, after selecting file, transformation begins. When it finishes, an information message appears, “Transformation is done”. The application asks for a file name to save the result as a FOM document, and displays the information message “OMT file saved”. Open the file, and if there are any unnecessary fields delete them. Or if “datatype” or “sharing” fields have to be changed, please edit them. Finally, produced file has the same format and content with the one which we produce manually.

Generating the FOM of Trajectory Simulation Federation by OWL2OMT tool is shown with the snapshots of OWL2OMT tool in the previous steps. These steps can be also figured as a sequence diagram as seen in Figure 30.

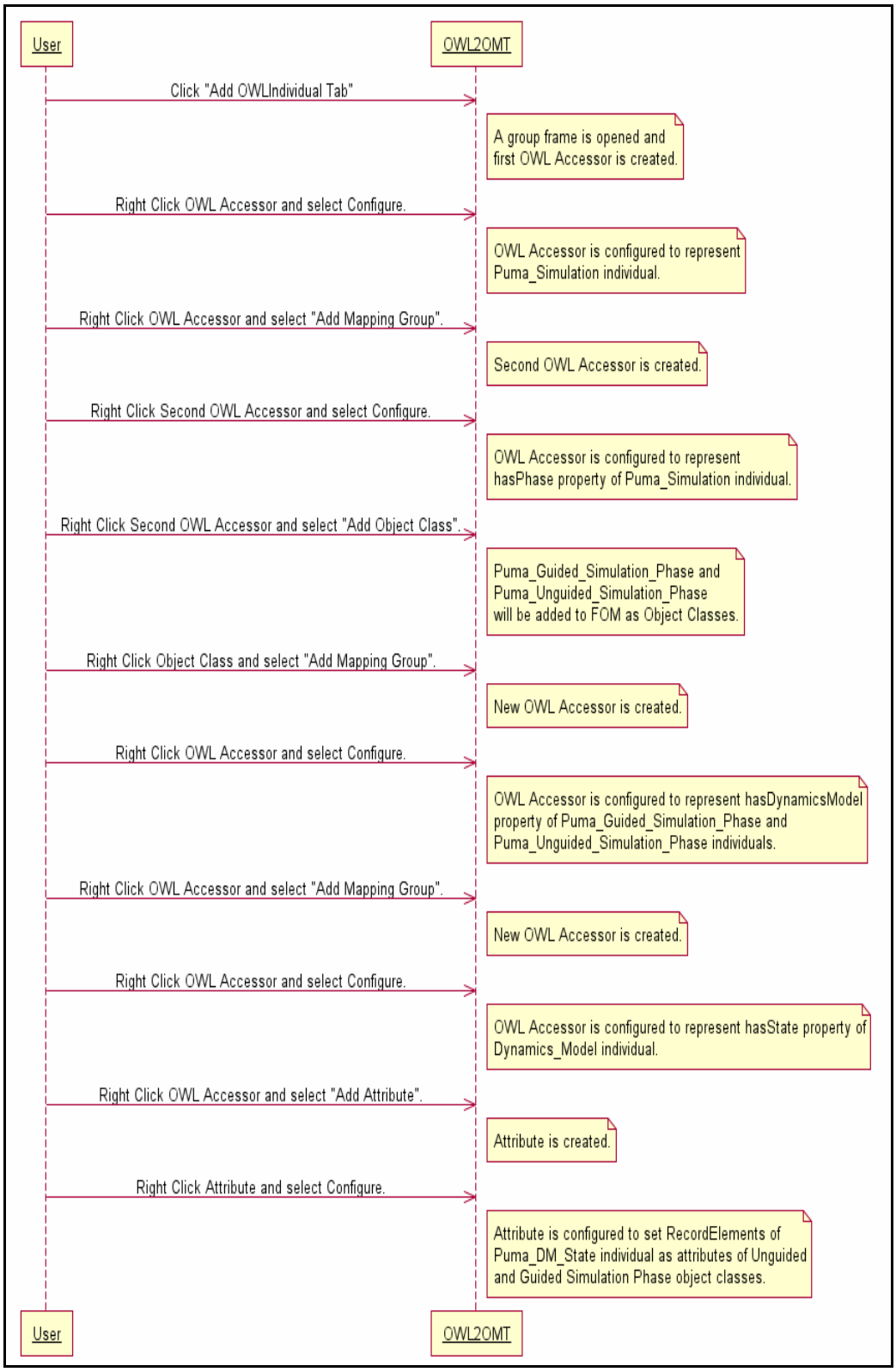


Figure 30 Generating FOM – Sequence Diagram

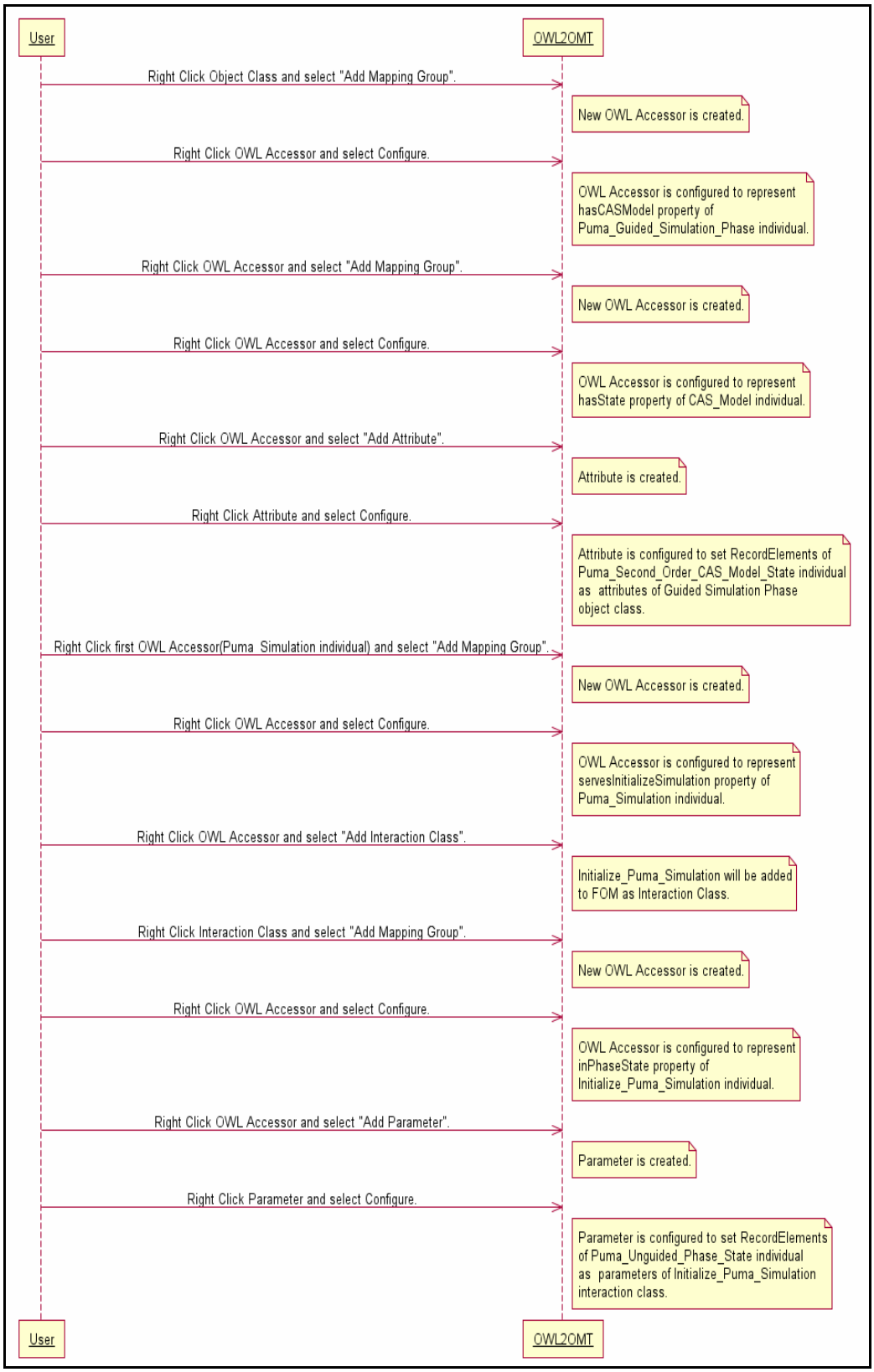


Figure 30 Generating FOM – Sequence Diagram (continued)

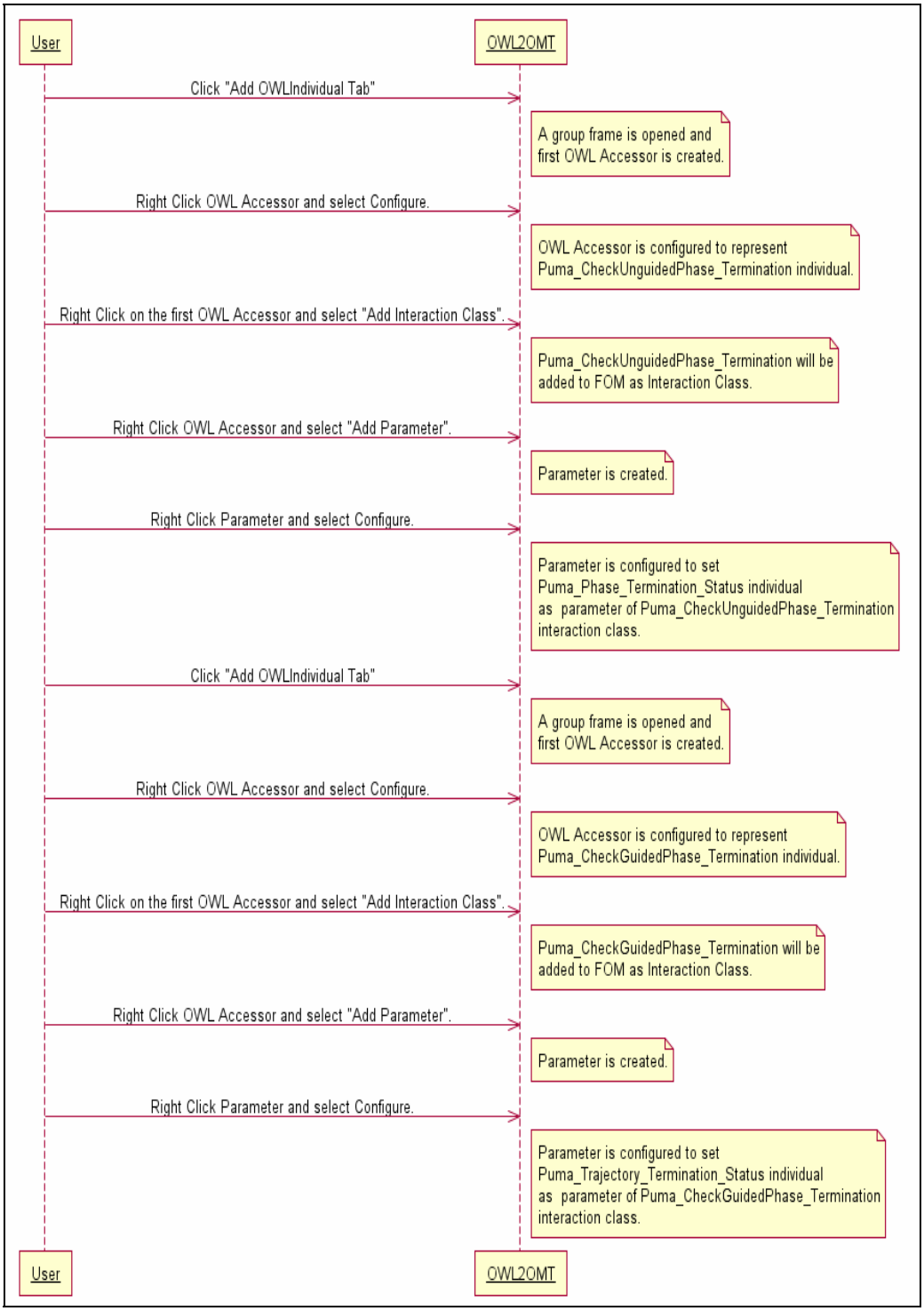


Figure 30 Generating FOM – Sequence Diagram (continued)

CHAPTER 4

FEDERATION OBJECT MODEL TO DISTRIBUTED SIMULATION

This chapter depicts the picture of the way from FOM to distributed simulation development. First, we will discuss the Trajectory Simulation Federation, the end product, then we will investigate federates and relations of them. As the third part, we will look into the development process of Simulation Application code, starting from the FOM. In the fourth part, we have the application, in other words Trajectory Simulation Federation. Finally, we will survey the last subject of this chapter, Execution of Federation.

4.1 Overview of Trajectory Simulation Federation

Trajectory Simulation Federation is a federation which is made up of two federates, Puma Federate and Exercise Manager Federate. There is an air to ground guided bomb, trajectory of this bomb during its flight is produced by Puma Federate and this trajectory information is followed by Exercise Manager Federate.

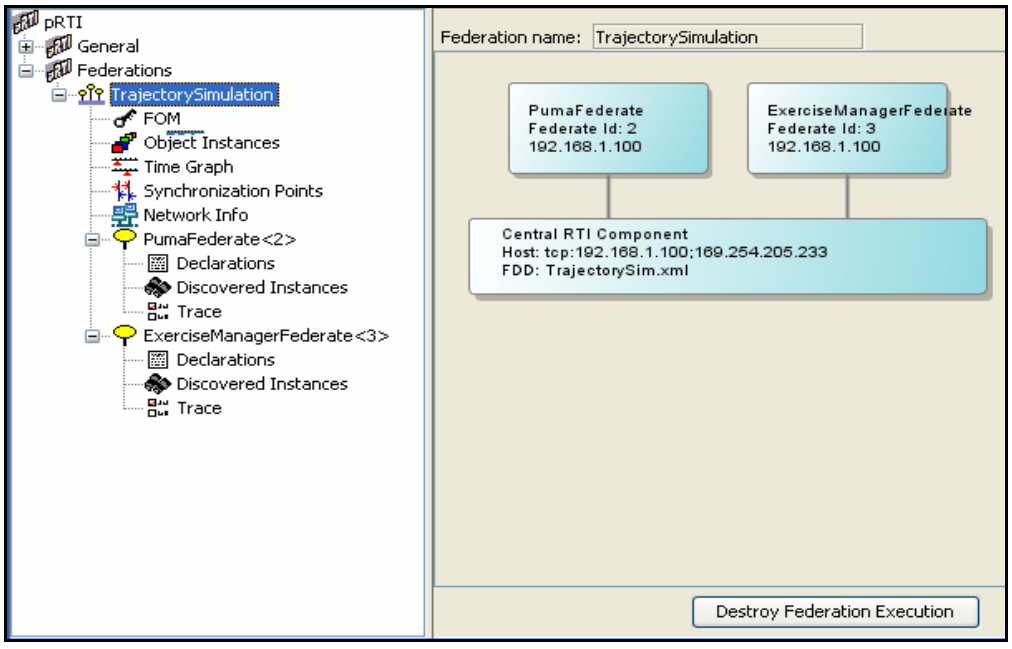


Figure 31 pRTI view of Federation

There are two phases of the bomb, so federates will produce and follow the trajectory for two different phases sequentially. First Unguided Phase, and then Guided Phase will be tracked. Phase transitions will be managed by the interactions, "Puma_Check_UnguidedPhase_ Termination" and "Puma_Check_GuidedPhase_Termination".

4.2 Federates and Relations

In the previous section, we have given a preface about "Federates and Relations". There are two federates, two HLA object classes and three HLA interaction classes which constitute the Trajectory Simulation Federation. Relation between federates and the role of interaction and object classes is shown in two Figures. Figure 32 represents the relations in the Unguided Phase of the flight.

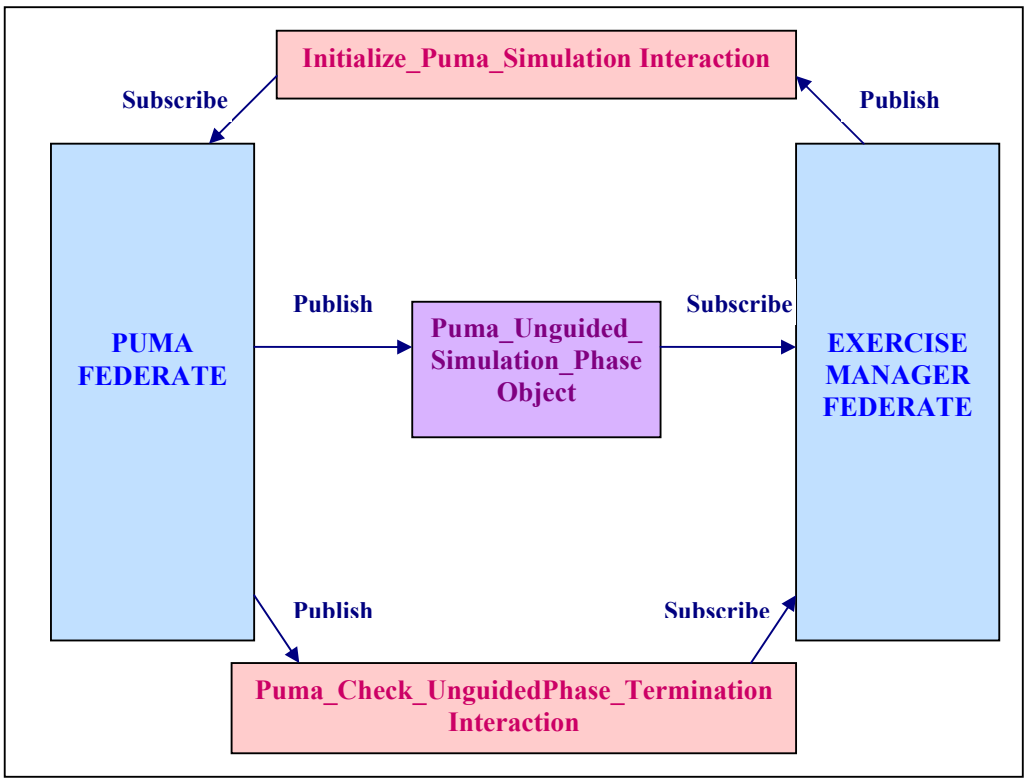


Figure 32 Federates and Relations in Unguided Phase

Exercise Manager federate publishes the Initialize_Puma_Simulation interaction and Puma federate subscribes to this interaction.When Puma federate receives that interaction, federate executes the Puma Trajectory Simulation executable with the parameters of Initialize_Puma_Simulation interaction. By executing Puma Trajectory Simulation, Puma federate obtains the trajectory information that Exercise Manager requests. These trajectory information depends on time and the phase of the munition. Puma federate publishes

Puma_Unguided_Simulation_Phase object and Exercise Manager federate subscribes to it. By this way Puma federate sends the trajectory information, Dynamic Model records for Unguided phase, to the Exercise Manager federate. When phase passes through Puma_Guided_Simulation_Phase, Puma federate sends an interaction Puma_Check_ UnguidedPhase_Termination interaction and Exercise Manager federate begins to receive the trajectory updates for Puma_Guided_Simulation_Phase object. At the end of the flight, Puma federate publishes another interaction Puma_Check_GuidedPhase_Termination, to inform Exercise Manager federate, that the bomb reaches to the target and the simulation is finished.

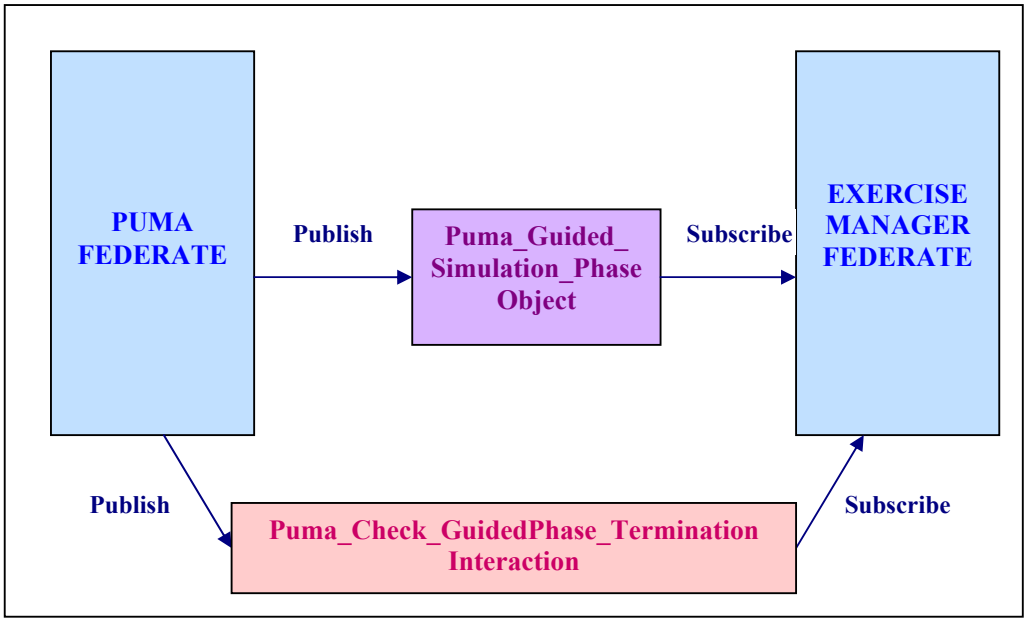


Figure 33 Federates and Relations in Guided Phase

4.3 Incorporating FOM into Simulation Application Code

Application development as a part of this thesis, consists of designing the federation, federates and writing the code of federates with Java programming in HLA in addition to using formerly developed applications such as Puma Trajectory Simulation. Puma Trajectory Simulation was developed using MATSIX as summarized in Section 2.3.

There are five Java classes; two of them are federate classes which exchange the objects and interactions derived from FOM. The other three classes are developed to manage time and attribute updates. These classes and their functionalities are summarized below.

Puma Class

Puma class represents the Puma Federate. It also includes Callback and ProvideAttributeValueUpdateCallback classes. ProvideAttributeValueUpdateCallback class extends from Callback class.

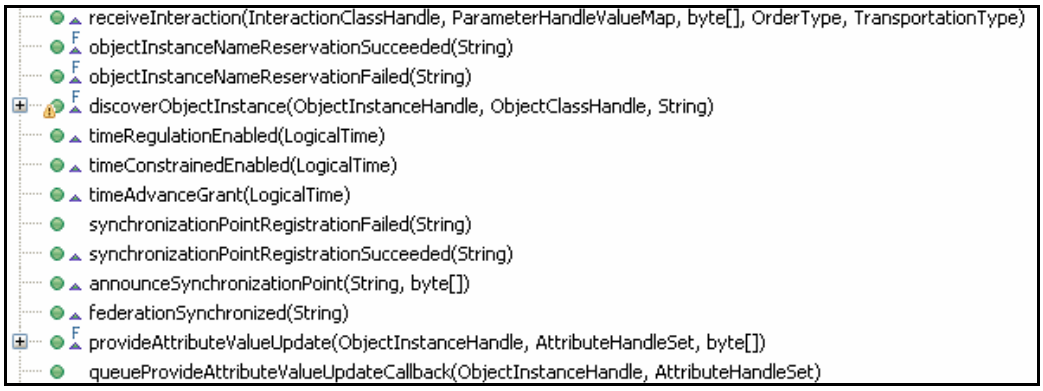


Figure 34 Functions in Puma Class

Functions of Puma Class are seen in Figure 34. Functions which have a blue mark besides their names, are imported from se.pitch.prti1516 library and overridden for supporting functionality of Puma Federate.

Exercise Manager Class

Exercise Manager Class represents the Exercise Manager Federate. It also includes Callback, ReflectAttributeValuesEvent and ExternalEvent classes. ReflectAttributeValuesEvent class extends from ExternalEvent class and ExternalEvent class extends from Callback class.

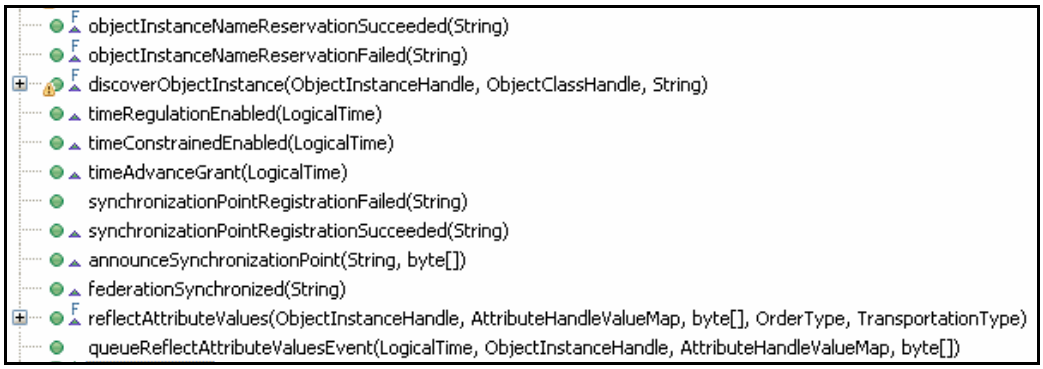


Figure 35 Functions in Exercise Manager Class

Functions of Exercise Manager Class are given in Figure 35. Functions which have a blue mark besides their names are imported from se.pitch.prti1516 library and overridden for supporting functionality of Exercise Manager Federate.

CallbackQueueProvide Class

This class is developed to manage the order of updating attributes, namely trajectory values. Methods included in this class are enqueue and dequeue methods. Callbacks which are in the type of ProvideAttributeValueUpdateCallback are enqueued and dequeued here.

CallbackQueueReflect Class

This class is developed to manage the order of updating object class attributes, namely trajectory values. Methods included in this class are enqueue and dequeue methods. Callbacks which are in the type of ReflectAttributeValuesEvent are enqueued and dequeued here.

Update Process of Attributes

Process to carry out an attribute update is as follows: Exercise Manager Federate sends a requestAttributeValueUpdate(imported from hla.rti1516 library) to Puma Federate and PumaFederate executes the provideAttributeValueUpdate method. When this method is executed at Puma Federate, RTI calls reflectAttributeValues method at Exercise Manager Federate. Federates have to check the queues in order to continue updating attributes. When Exercise Manager Federate requests for an update, execution of this federate is locked by dequeue method of CallbackQueueReflect class till it receives an update from Puma Federate. Puma Federate is locked by dequeue method of CallbackQueueProvide class till its time is advanced.

4.4 Execution of Federation: Trajectory Simulation Federation

In this section, we'll discuss the execution of the federation, by examining sequence diagrams. Execution process will be investigated with two figures. Figure 36 represents the first part;

- Exercise Manager Federate creates the federation and joins it.
- Puma Federate joins the federation.
- Exercise Manager Federate gets the object class handle of Puma_Unguided_Simulation_Phase object and gets the attribute handles of Puma_DM_State records.
- Puma Federate gets the object class handle of Puma_Unguided_Simulation_Phase object and gets the attribute handles of Puma_DM_State records.
- Puma Federate publishes attributes of Puma_Unguided_Simulation_Phase object.
- Exercise Manager Federate subscribes to attributes of Puma_Unguided_Simulation_Phase object.

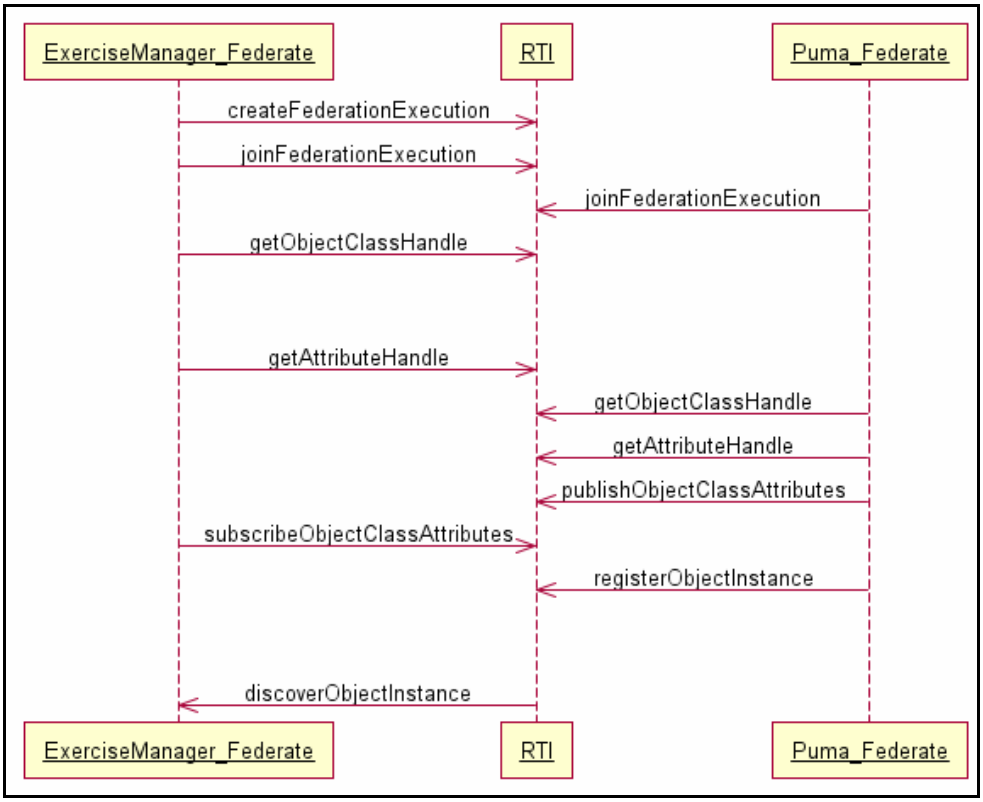


Figure 36 Execution of Federation - 1

- Puma Federate registers an instance of Puma_Unguided_Simulation_Phase object.
- Exercise Manager Federate discovers the object instance that Puma has registered.

Figure 37 presents the second part;

- Exercise Manager Federate publishes Initialize_Puma_Simulation interaction class.
- Puma Federate subscribes that interaction class.
- Exercise Manager Federate sends Initialize_Puma_Simulation interaction and Puma Federate receives it from RTI.
- When Puma Federate receives that interaction, the federate executes “Puma Trajectory Simulation” executable. At the end of execution of “Puma Trajectory Simulation”, Puma Federate is able to provide the trajectory values.
- Exercise Manager requests for attribute updates, to get the Puma_DM_State records.

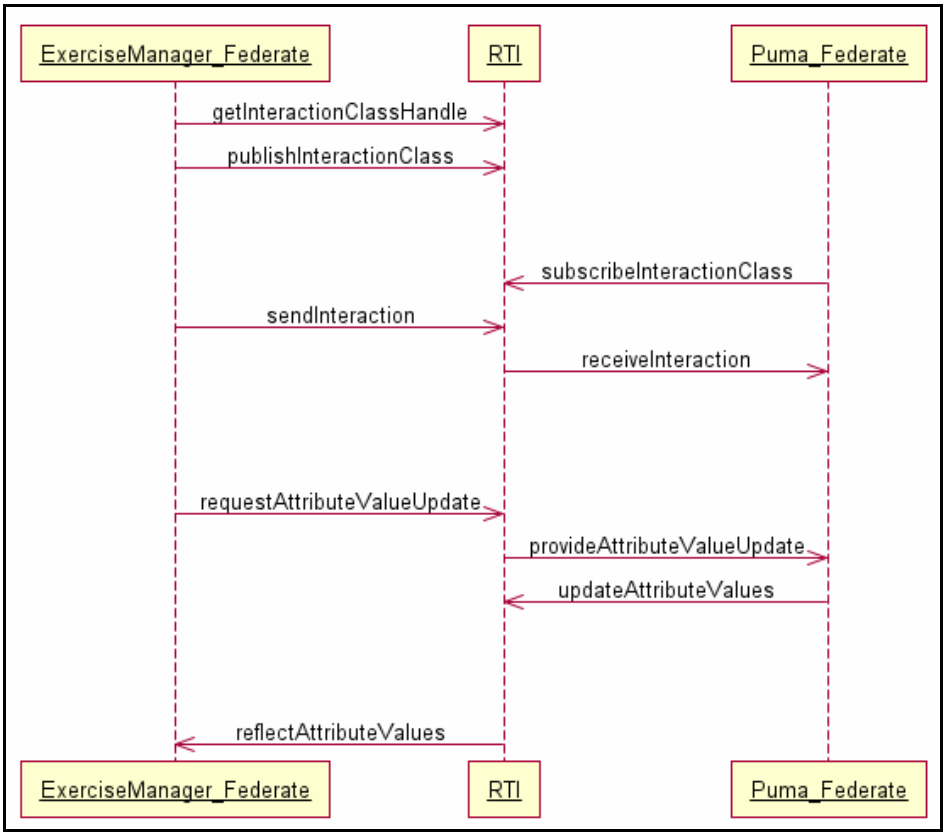


Figure 37 Execution of Federation – 2

- RTI calls provide attribute value update at Puma Federate and Puma Federate sends update for attribute values.
- RTI reflects the updated values to Exercise Manager Federate.
- Update – Reflect process continues to execute at every TimeAdvanceGrant.

These are the Puma_Unguided_Simulation_Phase part of the execution which provides the updates for Unguided Phase of the bomb.

As we mentioned earlier, when Puma_Unguided_Simulation_Phase finishes, Puma_Guided_Simulation_Phase starts. Puma Federate sends an interaction named Puma_Check_UnguidedPhase_Termination to inform Exercise Manager Federate about phase change. ExerciseManager subscribes to Puma_Guided_Simulation_Phase object and begins to receive attribute updates for Puma_Guided_Simulation_Phase from Puma Federate.

Time Management in Federation Execution

Time management in the federation is implemented with the functions enableTimeRegulation(), enableTimeConstrained() and timeAdvanceRequest().

ExerciseManager Federate requests an update for attribute values only once. Then in the time loop, advances its logical time, waits for receiving updates from Puma, when it receives a new update, advances its own logical time again. Puma Federate advances its own logical time whenever it sends the attribute updates to Exercise Manager Federate. Time management is summarized in Figure 38.

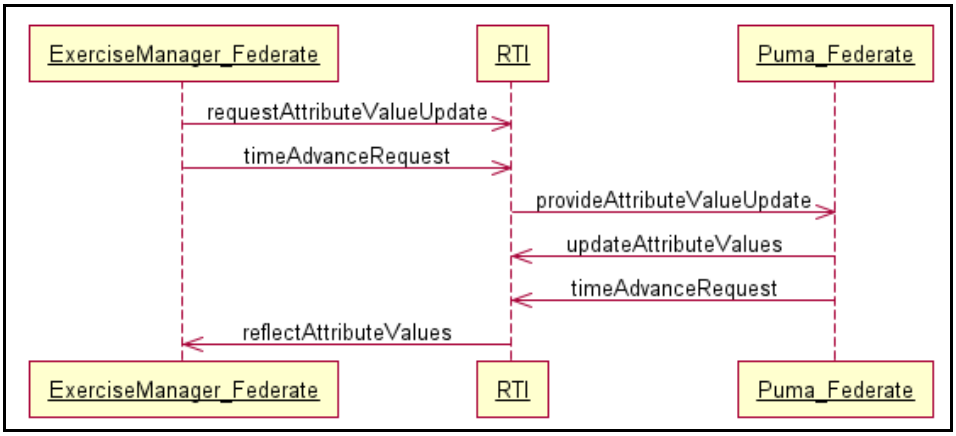


Figure 38 Time Management

CHAPTER 5

CONCLUSION

The achievement of the thesis is generation of HLA federates based on ontology. Ontology driven development of HLA federates offer methodological approach besides interoperability and composability. As a core part of this study, starting from analyzing TSONT, FOM rules are defined and FOM is developed by manually and by OWL2OMT. Secondary acquisition of this thesis is generating HLA federates using the FOM. Java based code is implemented to constitute federates, this part is named HLA wrapper.

The produced FOM file carries out the rules of the Trajectory Simulation Federation. These rules are vitally important for federates, because these rules define the communication way between the federates.

OWL2OMT is used to generate the FOM file besides generating manually. OWL2OMT provides a methodological approach for transformation from OWL to Object Model Template.

Puma Trajectory Simulation is the core calculation model and implementation part of the simulation that produces the trajectory information during the flight of the munition. Puma Trajectory Simulation is wrapped by HLA wrapper which is the Java based code of federates. Creating the federation, joining the federation, calling Puma Trajectory Simulation and tracking the trajectory by requesting/sending attribute updates and sending/receiving interactions composes the HLA wrapper

As a future work, it may be thought that to integrate these ontology driven federates with an open source flight simulator, such as Flightgear. Another work to do is elaborating features of OWL2OMT such as editing data type or sharing properties.

REFERENCES

[1] U.S. Department of Defense, 1995, Missile Flight Simulation, Part One Surface-to-Air Missiles, MIL-HDBK 1211.

[2] YILMAZ, L., (2007). Using Meta-Level Ontology Relations to Measure Conceptual Alignment and Interoperability of Simulation Models, Winter Simulation Conference 2007, 1090-1099.

[3] DAVIS, K. P. and ANDERSON A. R., (2003). Improving the Composability of Department of Defense Models and Simulations, RAND Technical Report available at http://www.rand.org/publications/MG/MG101/ (last accessed December 2009)

[4] DURAK, U, OĞUZTÜZÜN, H., and İDER, K., (2009). Ontology-Based Trajectory Simulation Framework, JCISE, March 2008

[5] ÖZDİKİŞ, Ö., DURAK,U., and OĞUZTÜZÜN, H.,(2009). OWL to UML : Transforming Domain Models to Framework Architectures. Summer Computer Simulation Conference 2009.

[6] ÖZDİKİŞ, Ö, DURAK, U., and OĞUZTÜZÜN, H., (2009). Tool Support for Transformation from an OWL Ontology to an HLA Object Model, SIMUTOOOLS 2010.

[7] NECHES, R., FIKES, R.E., FININ, T., GRUBER, T.R., SENATOR, T. and SWARTOUT, W.R. (1991). Enabling Technology for Knowledge Sharing, AI Magazine, 12 (3):36-56.

[8] GRUBER, R. (1993), A Translational Approach to Portable Ontology Specifications, Knowledge Acquisition, 5(2).

[9] MİZOGUCHİ, R. (2001), Ontological Engineering: Foundations of the Next Generation Knowledge Processing, Web Intelligence 2001, Maebashi City, Japan, 2001, as cited in [11].

[10] ARANGO, G.(1989), Domain Analysis: From Art to Engineering Discipline. In Proceedings of 5th International Workshop on Software Specification and Design, Pittsburgh, PA., as cited in [11].

[11] DURAK, U. (June 2007), ONTOLOGY BASED REUSE INFRASTRUCTURE FOR TRAJECTORY SIMULATION, PhD Thesis, METU

[12] WATT, D.A. (1990), Programming Language Concepts and Paradigms, Prentice Hall Series in Computer Science as cited in [11].

[13] FITZGERALD, J., LARSEN, P.G. (1998), Modelling Systems: Practical Tools and Techniques in Software Development, Cambridge University Press.

[14] GENNARI, J.H., MUSEN, M.A., FERGERSON, R.W., GROSSO, W.E., CRUZBY, M., ERIKSSON, H., NOY, N.F., and TU, S.W. (2003), The Evolution of Protégé: An Environment for Knowledge-Based Systems Development, International Journal of Human-Computer Studies, 58 (1):89-123.

[15] HLA Working Group of IEEE, (2000), IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules, New York, USA.

[16] TOLK A., TURNITSA C. (2006), Ontology Applied – Techniques employing Ontological Representation for M&S, Fall Simulation Interoperability Workshop, Simulation Interoperability Standards Organization, IEEE CS Press.

[17] TOLK A., TURNITSA C., DIALLO S., (2006), Ontological Implications of the Levels of Conceptual Interoperability Model. World Multi-Conference on Systematics, Cybernetics and Informatics (WMSCI) 2006, Orlando, FL as cited in [16].

[18] TURNITSA C. and TOLK A. (2006). "Battle Management Language: A Triangle with Five Sides," Paper 06SSIW-016, Spring Simulation Interoperability Workshop, Simulation Interoperability Standards Organization, IEEE CS Press, April 2006 as cited in [16].

[19] TOLK A., CIVINSKAS W., and DIALLO, S. (2006). An Application Extension for the Military Scenario Description Language. 2006 Spring Simulation Interoperability Workshop, Huntsville, AL, IEEE CS Press, 2006 as cited in [16].

[20] BHATT M., RAHAYU W., STERLING G. (2004). "sedOnto: A Web Enabled Ontology for Synthetic Environment Representation Based on the SEDRIS Specification, "2004 Fall Simulation Interoperability Workshop Proceedings, IEEE CS Press, 2004 as cited in [16].

[21] MILLER John A., BARAMIDZE Gregory T. , SHETH Amit P. , FISHWICK Paul A., (2004), Investigating Ontologies for Simulation Modeling, Proceedings of the 37th annual symposium on Simulation, p.55, April 18-22, 2004.

[22] MILLER J.A., BARAMIDZE G.T., SHETH A.P. , SILVER G.A., (2008),University of Georgia and FISHWICK P.A., University of Florida, Ontologies for Modeling and Simulation: An Initial Framework.

[23] LACY L., AVILES G., FRASER K., GERBER W., MULVEHILL A., GASKILL R. (2005), Experiences with using OWL in Military Applications, OWL: Experiences and Directions Workshop, Galway, Ireland, November 11-12, 2005.

[24] Pitch Technologies, (February 2005), pRTI 1516 User's Guide.

[25] DURAK U., OĞUZTÜZÜN H., KÖKSAL ALGIN C.F., ÖZDİKİŞ Ö. (2010), Towards Interoperable and Composable Trajectory Simulations: An Ontology Based Approach, submitted.

[26] HORRIDGE, M., KNUBLAUCH, H., RECTOR, A., STEVENS, R. and WROE, C., A (2007), Practical Guide to Building OWL Ontologies Using The Protégé – OWL Plugin and CO-ODE Tools Edition 1.0.