A MODEL TRANSFORMATION APPROACH FOR THE DEVELOPMENT OF HLA-BASED DISTRIBUTED SIMULATION SYSTEMS

Andrea D'Ambrogio, Giuseppe Iazeolla, Alessandra Pieroni
University of Roma TorVergata, Rome, Italy

Daniele Gianni
European Space Agency, Noordwijk, The Netherlands

Keywords: Distributed simulation system, High level architecture, HLA, Model driven development, MDA, Model transformation.

Abstract: The development of HLA-based distributed simulation systems requires a significant expertise and a considerable effort for the inherent complexity of the HLA standard. This paper introduces an automated approach for the development of HLA-based simulation systems of higher quality at largely reduced time, effort and cost. The proposed approach is founded on the use of model transformation techniques and relies on standards introduced by the Model Driven Architecture (MDA). The proposed approach takes as input a UML model of the system to be simulated and yields as output both an intermediate UML model and the final code of the HLA-based distributed simulation system.

1 INTRODUCTION

The most prominent standard for distributed simulation, *High Level Architecture (HLA)* (IEEE 1516, 2000), has been initially proposed to increase the reuse and the interoperability of simulation software, as a way of reducing the development effort of complex distributed simulation systems. After the release, however, HLA has failed to fulfill and anticipate the community needs in terms of semantic interoperability (Tolk & Muguira, 2003) and effortless development (Gianni, D'Ambrogio & Iazeolla, 2008)). A generalization of this latter concept can be envisioned by reducing the technical coding effort with the introduction of an automated approach to the development of HLA-based software systems.

Automating the derivation of software code from a high-level model specification has proven effective to cope with the increasing complexity of modern software systems. Asides from shortening software production time, an automated approach to software development brings other advantages. These are related to the *model abstraction level* for the system design, and to the increased reuse of software products. Providing a *model abstraction level* for the design and implementation of software eliminates the pure technical *effort* that the final implementation would require. In addition, this approach fosters the *reuse* of software products, such as software model patterns and software components, and thus also increases the quality of the final product. As a consequence of both these advantages, the software code can be obtained for various platforms with no extra effort.

Commercial software industries have fully embraced these considerations and have introduced standard technologies that support the automation of the entire development process. In particular, the Object Management Group has introduced *MDA* (*Model-Driven Architecture*) (OMG, 2003), a set of specifications addressing the issues of automating software production from *UML* (*Unified Modeling Language*) model descriptions of software systems (OMG, 2009). MDA supports the model-driven development of software-intensive systems through the transformation of *platform independent models*

D'Ambrogio A., Iazeolla G., Pieroni A. and Gianni D..
A MODEL TRANSFORMATION APPROACH FOR THE DEVELOPMENT OF HLA-BASED DISTRIBUTED SIMULATION SYSTEMS.
DOI: 10.5220/0003599001550160
In *Proceedings of 1st International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2011), pages 155-160
ISBN: 978-989-8425-78-2

(*PIMs*) to *platform specific models* (*PSMs*), executable components and applications.

As aforementioned, the HLA standard promotes interoperability and reusability of simulation components in different contexts and is based on the following concepts: federate, federation and RTI (Run Time Infrastructure). A *federate* is a simulation program and represents the unit of reuse within HLA. Federates communicate through shared object instances (i.e., permanent entities) and by interactions (i.e., messages). A *federation* is a distributed simulation execution composed of a set of federates. Finally, the *RTI* is a simulation-oriented middleware that coordinates the execution of federates. An HLA-based simulation system is described by use of a set of *Simulation Object Models (SOMs)* and a *Federation Object Model (FOM)*. Each SOM documents a single federate, while the FOM documents the entire federation.

Similarly to general purpose software, HLA-based distributed simulation systems are becoming increasingly more complex, and therefore a MDA-based approach for the production of such systems can be seen as an effective solution for supporting the development of distributed simulation systems of higher quality at largely reduced time and effort.

This paper introduces a model transformation approach that exploits MDA standards and techniques to automate the production of HLA-based distributed simulation systems.

The approach takes as input a UML model of a system to be simulated and produces as output the code of the corresponding HLA-based simulation system. To achieve such an objective, the model transformation approach introduces two transformation steps, the former to obtain an intermediate HLA-specific UML model from the initial UML model and the latter to yield the software code — ready for execution — of the HLA-based simulation system from the HLA-specific UML model.

In addition, the approach also introduces a HLA-based UML profile, i.e., a standard UML extension mechanism to specify the intermediate HLA-specific UML model by use of HLA-specific annotations.

The paper is organized as follows. Section 2 remarks this paper contribution compared to other state-of-the-art works. Section 3 presents the details of the proposed two-steps *model transformation approach*. Finally, Section 4 concludes the paper by illustrating the automatic derivation of the HLA code for the distributed simulation of a mobile SIM card management system.

2 RELATED WORK

Model specification technologies based on model-driven approaches are used to reduce the gap between the model specification and the distributed system implementation. These technologies do not impose any constraints on system modeling, and similarly obtain a reduction of development effort by automating the production of simulation code from a formal model specification. In this field, the contribution proposed in this paper can be compared to contributions introduced in (Tolk & Muguira, 2004), (Parr & Keith, 2003), (Jiménez, Galan & Gariía, 2006) and (El Haouzi, 2006).

The contribution in (Tolk & Muguira, 2004), differs from this paper approach in terms of the application strategy, the UML diagrams adopted and the state of implementation. Concerning the *application strategy*, the contribution proposes the creation of a specific domain for Modeling and Simulation (M&S) within MDA. Differently, this paper focuses on the application of MDA techniques to the production of simulation systems treated as general-purpose software systems. This means that, in case of simulation of a software system, the same approach can be adopted to eventually generate both the operational system and the simulation system from the same model specification. The set of *UML diagrams* adopted in (Tolk & Muguira, 2004) is wider and includes implementation diagrams. Conversely, this paper focuses on a narrower set of diagrams, including class diagrams and interaction diagrams. The *state of the implementation* of contribution in (Tolk & Muguira, 2004), however, is not complete in terms of both MDA compliance and software. Differently, this paper approach implements a MDA compliant process by introducing an HLA-specific UML profile and two transformations for deriving the simulation code from a UML model specification.

Similarly, the approach in (Parr & Keith, 2003) differs from this paper contribution in three ways: visual representation of specification model, MDA compliance and software technologies for code derivation. A *visualization tool* for representing HLA objects and for deriving the corresponding FOM and SOMs is introduced. Differently, this paper's approach relies on general-purpose visualization tools, which also prove to be more reliable and open to extensions. The *MDA compliance* achieved in (Parr & Keith, 2003) is lower because it does not provide either an HLA-based UML profile or the transformations

introduced in this paper. As regards *software technologies for code derivation*, the contribution in (Parr & Keith, 2003) adopts proprietary technologies, while this paper contribution is founded on standard languages for model transformation.

The contribution in (Jiménez, Galan & Gariía, 2006) also proposes a MDA-based development of HLA simulation systems. However, this contribution is limited to the definition of an initial UML profile for HLA.

Finally, the contribution in (El Haouzi, 2006) only outlines the main concepts behind the application of MDA techniques to the development of HLA systems.

3 MODEL TRANSFORMATION APPROACH

The development of HLA-based distributed simulation systems is defined by three phases: development of the federation conceptual model, design of the federation, and development of the federation. The *development of the federation conceptual model* cannot be automated because requires understanding and creativity. Differently, the *design* and the *development of the federation* from the conceptual model require only technical and mechanical skills, and therefore such phases can be automated. As a consequence, a model-driven design with an increased reuse of simulation components can be successfully applied. The former enables developers to cope with the increasing complexity of simulation model. The latter contributes to reduce the time required by validation and verification sessions, as simulation components yield increased reliability and accuracy. In addition, the adoption of a standard methodology, such as MDA, brings two other advantages. *First*, developers produce model specifications in standard UML, and therefore they need not be concerned with either the HLA modeling paradigm or any other non standard paradigm. *Second*, design and development phases can be carried out by use of standard tools for model definition and transformation (e.g., the many commercially available or freeware UML-based software development environments).

Figure 1 describes the model transformation approach, which is based on two steps. The *first step* consists of the PIM-to-PSM transformation that maps the PIM of the system to be simulated into the PSM of the HLA-based distributed simulation system. This step takes as input the UML profile of the HLA platform, HLA profile, which allows annotating standard UML elements with HLA-specific features. The *second step* takes as input the PSM obtained from the first step and yields as output the final code of the HLA-based distributed simulation system. This step requires the choice of a specific HLA implementation (e.g., Pitch, Portico, etc.) that provides the HLA services in a given programming language (e.g. Java or C++).

The following subsections describe the PIM-to-PSM transformation and the PSM-to-Code transformation, respectively. The details of the HLA profile can be found in (D'Ambrogio, Loprieno & Tiberia, 2009).

3.1 From PIM to PSM

The first step of the model transformation approach consists of the PIM-to-PSM transformation. The transformation rules are here presented informally, using natural language. Their formal specification has been specified in the *QVT* model transformation language (OMG, 2008).

To simplify the implementation of the transformation it is assumed that the input PIM is a UML model that consists of a use case diagram, a set of sequence diagrams for each use case and a class diagram built according to the MVC (Model-View-Controller) architectural pattern. This pattern structures the model description into *entity* classes

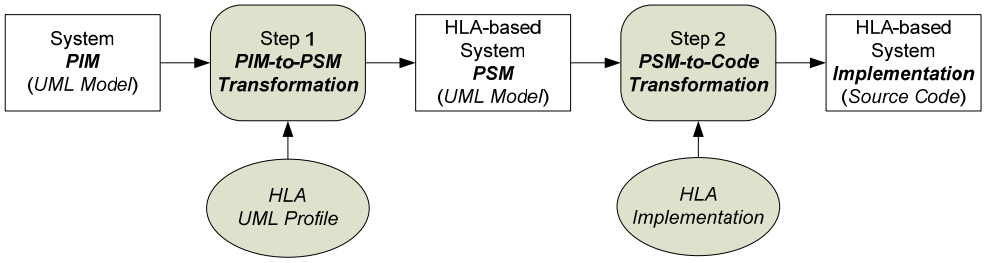

Figure 1: The two-step model transformation approach for the development of HLA-based systems.

(i.e., domain-specific data), *control* classes (i.e., business logic) and *boundary* classes (i.e., interfaces between system and actors).

This section illustrates a subset of the transformation rules that map the input UML model into the output HLA-based UML model stereotyped by use of the HLA profile. The subset includes the six rules described below.

Rule 1: Actor-Boundary to Federate

Each PIM actor-boundary couple of classes must be mapped into a PSM federate.

Within a PIM, a *boundary* class represents the communication interface between system and outside world. Similarly, the *actor* entity models the user submitting requests to the system. Differently, within the HLA-based PSM, the *boundary* concept is not present because *actor* entities are modeled within the distributed simulation system. Each actor entity in the PIM is always provided with a boundary interface, and therefore the actor–boundary couple is mapped into the smallest independent unit (i.e., *Federate*) within the HLA-based PSM.

Rule 2: Control to Federate

The control units of the set of use cases in the PIM must be mapped into a single federate in the PSM.

A PIM *control unit* represents a class that manages the flow of execution within a use case. Thus, in a PIM there are as many *control* classes as the number of use cases. Such PIM classes are mapped into a single HLA federate that manages the requests of all federates. Therefore, the set of PIM control classes is to be mapped into a single PSM class stereotyped as *Federate*.

Rule 3: Entity to ObjectClass

Each PIM entity class must be mapped into a PSM object class.

A PIM entity models a persistent object used to exchange data between two or more actors. Such a concept is naturally mapped into the HLA Object class concept, and thus each PIM entity class is mapped to a PSM class stereotyped as *ObjectClass*.

Rule 4: Associated Entities to Federate

A set of associated PIM entities must be mapped into a single PSM federate.

PIM entities can be grouped to form a set of entities with a *main* entity (e.g., an entity that models a DB table with a set of foreign keys) and a number of associated entities (e.g., the set of tables referenced by the foreign keys). In such a case, the set of entities is mapped into a single PSM class stereotyped as *Federate*, by applying Rule 3 to the main entity grouping all the entities (e.g., a single database table rather than a set of database tables associated by use of foreign keys).

Rule 5: Actor-Boundary Messages to Self Messages

Each sequence diagram message exchanged between a PIM actor object and a PIM boundary object must be mapped to a self message of the corresponding PSM federate.

This rule is applied to PIM sequence diagrams to obtain the corresponding PSM sequence diagram and takes into account the transformation specified by the forenamed Rule 1.

Rule 6: Boundary-Control messages to RTI messages

Each sequence diagram message exchanged between a PIM boundary object and a control object must be mapped to a message between the corresponding PSM federate and the RTI executive, and vice versa.

This rule is applied to PIM sequence diagrams to obtain the corresponding PSM sequence diagram and takes into account both the transformation specified by the forenamed Rule 1 and the use of the RTI middleware for carrying out the interactions between the federates of a HLA-based distributed simulation system.

3.2 From PSM to Code

The second, and last, step of the proposed model transformation approach for the development of HLA-based systems consists of the PSM-to-Code transformation.

The transformation rules are here presented informally, using natural language. Their formal specification has been specified in the *Xpand* model-to-text transformation language (EMF, n.d.).

This section illustrates the three rules that map the PSM model, consisting of class and sequence diagrams, into the corresponding code of the HLA-based distributed simulation system.

Rule 1: PSM Federate to Java Federate and Federate Ambassador Classes

Each Federate PSM entity is mapped into two Java classes, one for the Federate code and one for the Federate Ambassador code.

The implementation of a HLA federate typically consists of three portions of statements: internal processing statements, RTI service statements, and RTI-initiated statements, because the RTI operates in an asynchronous multi-threading manner.

Therefore, each Federate PSM entity is mapped into a Java class that contains the internal and RTI service statements and a FederateAmbassador class that contains the RTI-initiated statements.

Rule 2: PSM Object Class to FOM Object Class

Each PSM Object Class entity is mapped into a FOM Object Class.

In the PSM, an Object Class represents a system persistent entity that does not incorporate any behavioral logic. Moreover, a persistent entity is also used to carry out a data exchange mean among different PSM Federates entities. The analogy with FOM Objects is thus immediate.

Rule 3: PSM Interaction Class to FOM Interaction Class

Each PSM Interaction Class entity is transformed in a FOM Interaction Class.

A PSM Interaction Class models an instantaneous communication between two or more Federate entities. Similarly to the PSM Object Class, the Interaction Class does not incorporate any behavioral logic. As a consequence, mapping a PSM Interaction class into a corresponding FOM class is immediate.

4 EXAMPLE APPLICATION

This section describes the application of the model transformation approach to the production of a simulation program for a prepaid SIM card management system (D'Ambrogio, Loprieno & Tiberia, 2009). The considered scenarios include three actors: User, Operator, and Bank. The *User* interacts with the system to recharge the SIM credit through a *Bank* operation, to update the SIM plan, to buy new offers, and to enquire for assistance by sending a request to an *Operator* of the phone company. From this summary of use cases, the PIM of the card management system is first derived in terms of an UML class diagram and a set of UML sequence diagrams for each of the above described use cases, as shown in Figure 2.

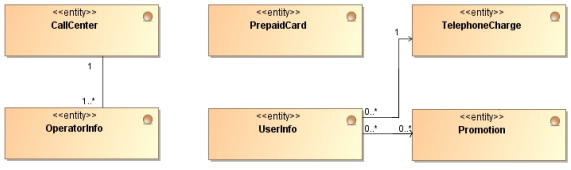

Figure 2: Entity classes for the example application.

The diagram consists of five classes stereotyped as <<entity>>: CallCenter, Operator, Prepaid Card, TelephoneRecharge, UserInfo, Promotion.

A *CallCenter* instance represents a phone company customer service and is associated to the set of *OperatorInfo* instances that describe the details of the phone company operators. A *PrepaidCard* instance describes a user SIM card. A *TelephoneRecharge* instance models the SIM card credit recharge and is associated to a *UserInfo* instance. A *UserInfo* instance might also be associated to the set of promotions subscribed by the user, as modeled by use of *Promotion* instances.

Sequence diagrams describing the interactions between class instances are defined for each use case.

The PIM-to-PSM transformation of Section 4.2 can be applied to the above PIM to obtain the HLA-based PSM for the SIM card management system.

The PSM is extended by use of the HLA profile and thus its elements are grouped in the following main packages: Federates, Interactions, Object Classes, Publish and Subscribe Diagrams and Behaviour Diagrams.

The *Federates* package includes the components stereotyped as <<Federate>>. Such components represent the federates of the HLA-based distributed simulation system, i.e.: CallCenterServer, CustomersServer, NetworkServicesServer, Operator, PrepaidCardsServer and User.

The *Interactions* package includes the components stereotyped as <<InteractionClass>> and the attributes stereotyped as <<InteractionParameter>>, which represent the interaction among federates and the interaction parameters, respectively. The package contains the following interaction classes: Message, Message_usr_srv, Message_usr_operator, Message_usr_disk.

The *Message* interaction class defines the generic interaction in terms of interaction parameters as type, sender, destination, timestamp, and so forth. The *Message_usr_srv* interaction class specializes the generic interaction for the communication between the User and the NetworkServiceServer federates. The *Message_usr_operator* interaction class specializes the generic interaction for the communication between the User and the Operator federates, while the *Message_srv_disk* interaction class specializes the generic interaction for the communication between the NetworkServiceServer federate and the PrepaidCardsServer, the CustomersServer and the CallCenterServer federates.

The *Object Classes* package includes the set of elements that represent the data exchanged by federates, i.e., classes stereotyped as <<ObjectClass>> and attributes stereotyped as <<ObjectAttribute>>. Figure 3 illustrates the hierarchy of object classes for the example application.

The *Publish/Subscribe Diagrams* package includes the set of diagrams representing the publish/subscribe specification of federates. Such specification enables the communication among federates within a federation.

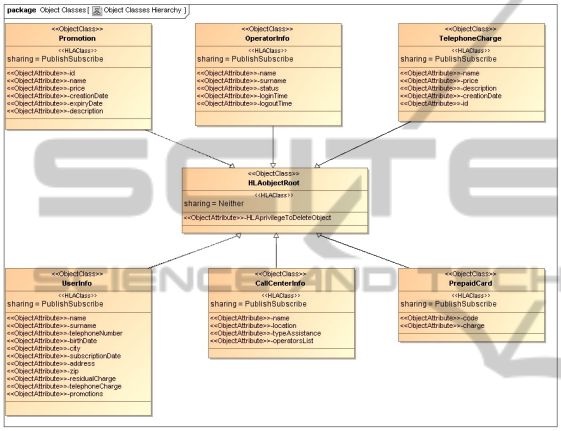

Figure 3: Hierarchy of PSM object classes.

The *Behavior Diagrams* package describes the federate behavior by use of a sequence diagram for each federate in the Federates package. The set of sequence diagrams is obtained by applying Rule 5 and Rule 6 of Section 3.1, plus a set of initialization and termination interactions that are automatically included for each federate.

The obtained PSM is finally used to derive the code of the HLA-based distributed simulation system, by applying to the PSM the set of rules described in Section 3.2. The transformation rules generate both the Java code of federates and the XML representation of the FOM. The Java code of federates is compliant to the Portico Open Source HLA implementation, while the FOM is coded according to the standard DTD for HLA FOMs.

The so obtained Java classes include the implementation of both initialization/termination methods and the `run()` method for each federate. Then, the set of federates can be allocated onto a set of possibly remote hosts for the distributed execution of the simulation system. This last activity may in turn be automated by introducing at PSM level a UML deployment diagram that describes the allocation of federate components onto the processing nodes of the execution infrastructure.

5 CONCLUSIONS

This paper has proposed a model transformation approach that derives the code of an HLA-based simulation system from a model specification of the system to be simulated. The approach has introduced a UML profile for HLA modeling and two transformations that automatically map the source UML system model into a HLA-specific model and eventually into the source code.

REFERENCES

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

Tolk, A., and Muguira, J. A., 2003. The Levels of Conceptual Interoperability Model (LCIM), *Proc. of the 2003 Fall Simulation Interoperability Workshop*.

Gianni, D., D'Ambrogio, A., and Iazeolla, G., 2008. A Layered Architecture for the Model-driven Development of Distributed Simulators, *Proc. SIMUTOOLS08, Marseille, France, March, 2008*.

OMG, 2003. *MDA Guide*, version 1.0.1.

OMG, 2009. *Unified Modelling Language*, version 2.2.

OMG, 2008. *MOF 2.0 QVT Specification*, version 1.0.

Tolk, A., and Muguira J. A., 2004. M&S within the model driven architecture, *Proc. of the Interservice/Industry Training, Simulation, and Education (I/ITSEC) Conferenc*e, 2004.

Parr, S., and Keith, R., 2003. The Next Step – Applying the Model Driven Architecture to HLA, *Proceedings of the 2003 Spring Simulation Interoperability Workshop*.

Jiménez, P., Galan, S., Gariía, D., 2006. Spanish HLA abstraction layer: towards a higher interoperability model for national, *Proc. of the European Simulation Interoperability Workshop*.

El Haouzi, H., 2006. Models Simulation and Interoperability using MDA and HLA, *IFAC/IFIP International conference on Interoperability for Enterprise Applications and Software (I-ESA'2006)*.

EMF, N. D., Eclipse Modeling Framework Project, <http://www.eclipse.org/modeling/emf>.

A. D'Ambrogio, S. Loprieno, G. Tiberia, 2009. *A Model-driven Approach for the Development of HLA-based Distributed Simulation Systems*, Technical Report RI.01.09, Software Engineering Lab, University of Roma TorVergata, Roma, Italy.