A Model-driven Framework for Distributed Simulation of Autonomous Systems

Paolo Bocciarelli
University of Rome Tor Vergata
53 PUBLICATIONS 605 CITATIONS

Andrea D'Ambrogio
University of Rome Tor Vergata
163 PUBLICATIONS 1,463 CITATIONS

Andrea Giglio
University of Rome Tor Vergata
27 PUBLICATIONS 284 CITATIONS

A Model-driven Framework for Distributed Simulation of Autonomous Systems

Paolo Bocciarelli, Andrea D'Ambrogio, Andrea Giglio and Emiliano Paglia
Dept. of Enterprise Engineering, University of Rome "Tor Vergata"
Rome, Italy
{paolo.bocciarelli,dambro,andrea.giglio,emiliano.paglia}@uniroma2.it

ABSTRACT

The adoption of systems with autonomous capabilities is becoming more and more relevant in many real-world operational scenarios, in which risky operations have to be carried out (e.g., a military battlefield or a search-and-rescue operation). In this context, innovative approaches should be introduced at design time to ensure that the system will achieve the mission objectives at operation time. To this purpose, distributed simulation techniques have shown to be effective to deal with the inherent complexity of the environment to be simulated, which generally includes several interacting entities. Unfortunately, currently available distributed simulation standards, such as HLA (High Level Architecture), require a non-negligible effort and significant skills in terms of both simulation methodologies and related implementation technologies. In this respect, this paper focuses on the simulation-based analysis of systems with autonomous capabilities and introduces a model-driven approach to support the automated generation of HLA-based distributed simulations. The proposed approach is founded on the use of model transformation techniques and allows system designers to carry out a timely and cost-effective simulation-based analysis of the operational system without being required to own specific distributed simulation skills

Author Keywords

distributed simulation; autonomous systems; model-driven development; MDA.

ACM Classification Keywords

I.6.5 SIMULATION AND MODELING: Model Development.

INTRODUCTION

Planning and executing operations in complex and risky scenarios (e.g., a military battlefield or a search-and-rescue operation) is a challenging issue that usually involves the coordination of various resources and personnel (e.g., different armed forces or search and rescue squads with multifaceted skills). The use of complex systems with autonomous capabilities, in other words systems that include autonomous resources, such as UAVs (Unmanned Aerial Vehicles), is becoming more and more relevant to the successful execution of such operations.

TMS/DEVS 2015 April 13-16, 2015, Alexandria, VA

The analysis and design of *autonomous systems* (ASs) requires the adoption of innovative techniques to assess whether the system will achieve the mission objectives at operation time. In this respect, simulation-based approaches can be effectively used from the early stages of the development process to predict the behavior of the system. The use of a simulation-based approach allows to analyze a system at design time, in order to asses whether or not it meets both functional and performance requirements and, ultimately, to cut or reduce the cost of experimental prototypes.

In order to effectively deal with the inherent complexity of the environment in which ASs are usually executed through the interaction of several entities, distributed simulation (DS) approaches could be conveniently introduced [4]. DS can be used to enact a scalable way to simulate a complex system by partitioning the overall simulation into a set of simulation components, each executed onto an independent host [8].

Nevertheless, the use of DS-based approaches is often limited by the non negligible effort and the significant skills required to make use of DS frameworks and environments, such as the HLA (High Level Architecture) framework and the related implementation technologies.

To overcome such limitations, this work proposes a method to support the automated generation of HLA-based distributed simulations starting from system descriptions specified by use of SysML (Systems Modeling Language), the UML-based general purpose modeling language for systems engineering [15].

The proposed method is carried out according to principles and standards introduced in the model-driven engineering field and is specifically founded on the Model Driven Architecture (MDA), the Object Management Group's incarnation of model-driven engineering principles [12].

The model-driven method focuses on model-to-model and model-to-text transformations that have been specified in QVT (Query/View/Transformation) [13] and MOFM2T (MOF Model to Text) [14] languages, respectively. The availability of such automated transformations allows system designers to derive the executable HLA-based simulation code with no extra effort and without being required to own specific DS skills, as shown by use of an example application.

The method makes use of various SysML/UML models that are created during the development of a distributed simulation. Each model represents the system from a specific point of view and at a given level of abstraction. SysML/UML models are thus used both to *represent* the system under a certain perspective and to embody the details required *to support the model transformations* upon which the proposed method has been based.

The proposed method also exploits two UML profiles, or modeling extensions, namely *SysML4HLA profile* and *HLA profile* [2]. The first one is used to annotate a SysML-based system specification in order to drive the automated generation of the HLA-based UML model. The second one is used to annotate a UML diagram in order to represent HLA-based details and drive the automated generation of the Java/HLA implementation code.

The rest of this work is organized as follows: a literature review that focuses on model-driven development approaches for developing both simulation systems and autonomous systems is first discussed. Then, the rationale of the proposed model-driven method is outlined. A running example is then introduced to give an insight into the method application and also to provide various implementation details. Finally, the paper conclusions are summarized.

RELATED WORK

This section reviews the existing literature dealing with the use of model-driven approaches in the development of both simulation systems and autonomous systems.

As regards the issue of implementing (or supporting the implementation of) simulation systems, several contributions can be found in literature that apply a model-driven paradigm in the modeling and simulation domain, such as [18, 5, 16, 11, 9].

In [18] the use of a model-driven paradigm in the M&S application domain, is proposed. This contribution 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 [18] is wider and includes implementation diagrams. Conversely, this paper focuses on a narrower set of UML diagrams annotated with stereotypes provided by SysML profile. The implementation of the proposed method is not complete in terms of both MDA compliance and tools. Differently, this paper approach implements an MDA compliant process by introducing two UML profiles and a set of model transformations for generating the simulation code starting from a SysML-based system specification.

In [5] a model-driven framework for modeling and simulation, denoted as MDD4MS, is presented. The paper illustrates both the metamodels and model transformations in the framework and a tool architecture implemented by use of an Eclipse-based prototype implementation. The authors discuss the advantages of MDD4MS and its applicability to the DEVS-based simulation of business processes specified in BPMN, without mentioning the applicability to HLA-based distributed simulation (even though the Distributed Simulation Object Library is used to provide the simulation and execution functionalities).

In [16] the application of a model-driven approach is introduced to support the development of simulation systems. The paper also underlines how HLA could benefit from the MDA and summarizes the most relevant technical requirements needed to integrate HLA and MDA. This contribution differs from this work along three different directions: 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 (Federation Object Model) and SOMs (Simulation Object Models) is introduced. Differently, this paper's approach relies on general-purpose visualization tools, which also prove to be more reliable and open to extensions. As regards software technologies for code derivation, the contribution in [16] adopts proprietary technologies, while this paper contribution is founded on OMG standard languages such as QVT.

In [11] an MDA-based development of HLA simulation systems, is also proposed. Such a contribution is limited to the definition of an initial UML profile for HLA and, differently from the contribution proposed in this paper, does not take into consideration the application of the profile for the implementation of the simulation system.

Finally, in [9] the main concepts behind the application of MDA techniques to the development of HLA systems are outlined. Such a contribution is limited to a theoretical discussion about the use of MDA-based techniques in HLA domain. Differently, this work proposes a model-driven approach to reduce the gap between the model specification and the distributed system implementation. An example application is also discussed, in order to show how the application of the proposed method allows to reduce the simulation development effort by automating the production of Java/HLA code from an initial UML-based system specification.

The use of model-driven approaches in the development of autonomous systems is addressed in [3] and [1].

In [3] a modeling approach is introduced to address the verification of autonomous systems through bidirectional model transformations, while in [1] MDA is only cited as a facility supported by UML/SysML modeling languages in the development of multi-agent systems. Both contributions deal with the use of model-driven approaches in the autonomous systems field at a level of abstraction higher than the one provided by this paper contribution, without concretely giving any implementation details.

OVERVIEW OF THE MODEL-DRIVEN METHOD

This section describes the rationale of the model-driven method for carrying-out a simulation-based analysis of autonomous systems. The method, which is outlined in Figure 1, is based on model-driven principles and standards and exploits the SysML4HLA profile and the HLA profile [2] to support both the modeling and the implementation of the HLA-based distributed simulation of the AS under study.

The system development process is concerned with two different domains. On the one hand, it is related to the system design and implementation. In this domain system engineers are not concerned with details regarding the simulation model, and they are strictly focused on the specification of a UML-based system design model, starting from the system requirements. On the other hand, the development process also addresses the simulation development and its execution, in order to predict the system behavior. In this domain, simulation engineers contribute to support the system development process by introducing simulation techniques to allow an early evaluation of the AS under study. In this respect, the proposed method aims at supporting both system and simulation engineers.

At the first step, the AS is initially specified in terms of SysML diagrams (e.g., block definition diagrams, sequence diagrams, etc.). According to model-driven terminology [12] such a model constitutes the platform independent model (PIM) of the system.

At the second step, the SysML4HLA profile is used to annotate the PIM in order to enrich such a model with information needed to derive the HLA simulation model. Specifically, the HLA profile allows to specify both how the system has to be partitioned in terms of federation/federates and how system model elements have to be mapped to HLA model elements such as *object class* and *interaction class*.

The third step takes as input the marked PIM and the HLA profile, and carry out the `SysML-to-HLA` *model-to-model* transformation, in order to automatically obtain the UML model that represents the HLA simulation model, i.e., a UML model annotated with the stereotypes provided by the HLA profile. According to model-driven terminology, such a model constitutes the platform specific model (PSM).

Finally, at the fourth step, the Java/HLA-based code of the simulation model is generated by use of the `HLA-to-Code` *model-to-text* transformation.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, C++, etc.)[1].

The following section gives a detailed view of each step by use of a running example.

METHOD IMPLEMENTATION AND APPLICATION

This section aims at giving a detailed view of the model-driven method introduced in the previous section. By use of an example application this section also shows how the proposed model-driven method can be effectively applied, in order to support the simulation of a system by generating the HLA/Java code, starting from its SysML specification. The proposed example application deals with the development of an autonomous *surveillance and engagement system* (SES).

With reference to Figure 1, the following subsections describe the several steps of the method.

Autonomous System Specification

The first step includes the definition of the SysML model of the AS under study. For the sake of brevity, this example only takes into account the the *fly-to-target* operation which consists of the following steps: the UAV Pilot specifies the UAV target by use of the ground station, then the UAV takes off and then flies to the target. When the target is reached, the UAV waits for the Pilot to activate the camera. The example focuses on the diagrams needed to derive the HLA simulation model and, consequently, the code of the distributed simulation.

Specifically, the model includes the following diagrams:

- *Block Definition Diagrams (BDDs)* and *Internal Block Diagrams (IBDs)*, to specify the structural view of the system. The diagrams show the UAV components (e.g., Ground Station, UAV, Weapon System, etc.) and their relationships;

- A set of *Sequence Diagrams (SDs)*, to specify the behavioral view of the system. Such diagrams describe the interactions between system components.

Federation Specification

At the second step, the SysML model is refined and annotated with stereotypes provided by the SysML4HLA profile. This step, which constitutes a PIM marking, adds to the SysML model the information needed to map each SysML domain element to the corresponding HLA domain element and makes the PIM ready to be automatically processed by the model transformations at the next steps. Figures 2, 3 and 4 show the BDD, the IBD and one SD for the example application, respectively.

Federation Design

The SysML model annotated with the SysML4HLA profile is given as input to the `SysML-to-HLA` *model-to-model* transformation to generate the HLA-based UML model, appropriately annotated by use of the HLA profile.

Before discussing the structure of the HLA-based simulation model, the `SysML-to-HLA` transformation is introduced in the next subsection.

SysML-to-HLA Transformation

The `SysML-to-HLA` model transformation takes as input the SysML model representing the system that is going to be simulated and yields as output the UML model that specifies the HLA-based distributed simulation. The input model is annotated with stereotypes provided by the SysML4HLA

[1]It should be noted that the implementation of `HLA-to-Code` transformation provided in this work makes use of Pitch and Java. Nevertheless, the model-driven approach at the basis of the proposed method allows to use different HLA implementations or programming languages. All such cases can be easily dealt with by revising the specification of the `HLA-to-Code` model-to-text transformation.

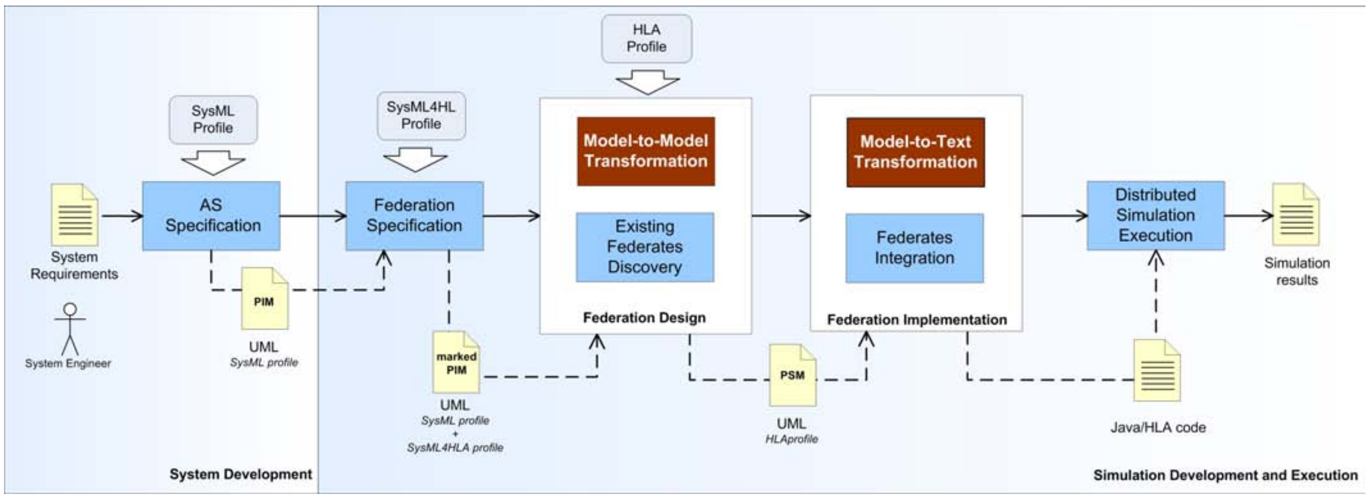

Figure 1. Model-driven method to automate the generation of Java/HLA code

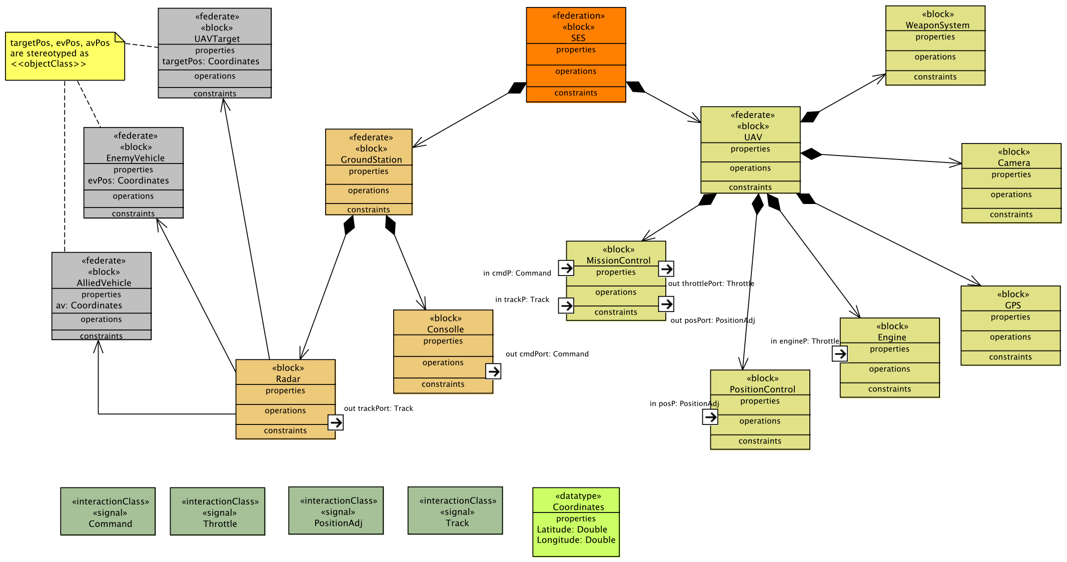
targetPos, evPos, avPos are stereotyped as <<objectClass>>
«federate» «block» UAVTarget properties targetPos: Coordinates operations constraints
«federation» «block» SES properties operations constraints
«block» WeaponSystem properties operations constraints
«federate» «block» EnemyVehicle properties evPos: Coordinates operations constraints
«federate» «block» GroundStation properties operations constraints
«federate» «block» UAV properties operations constraints
«block» Camera properties operations constraints
«block» MissionControl properties operations constraints
in cmdP: Command
in trackP: Track
out throttlePort: Throttle
out posPort: PositionAdj
«federate» «block» AlliedVehicle properties av: Coordinates operations constraints
«block» Consolle properties operations constraints
out cmdPort: Command
«block» GPS properties operations constraints
«block» Engine properties operations constraints
in engineP: Throttle
«block» PositionControl properties operations constraints
in posP: PositionAdj
«block» Radar properties operations constraints
out trackPort: Track
«interactionClass» «signal» Command
«interactionClass» «signal» Throttle
«interactionClass» «signal» PositionAdj
«interactionClass» «signal» Track
«datatype» Coordinates properties Latitude: Double Longitude: Double

Figure 2. Source model: Block Definition Diagram

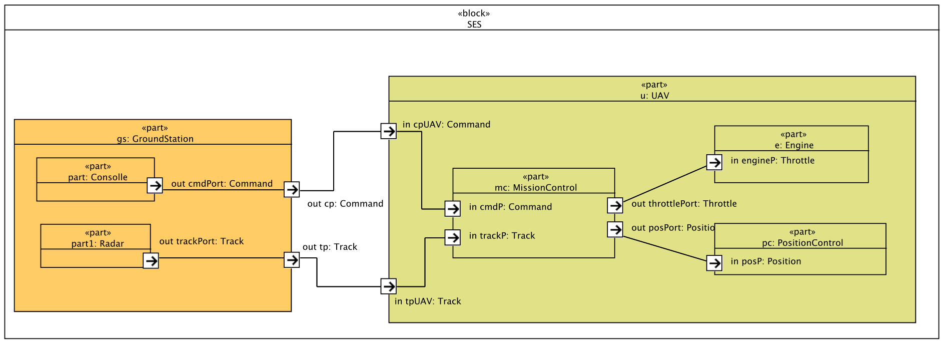

Figure 3. Source model: Internal Block Diagram (SES block)

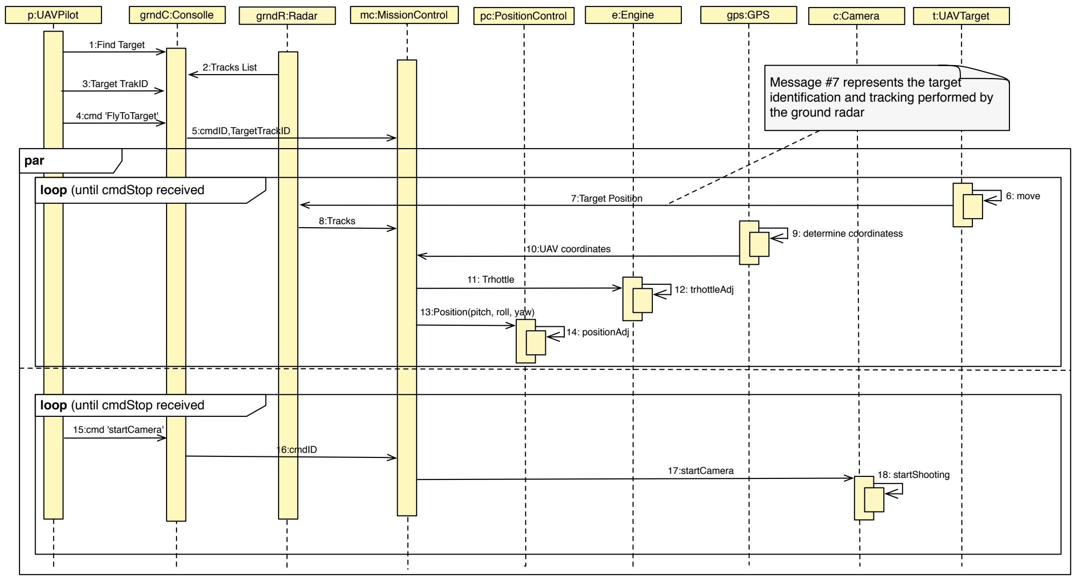

Figure 4. Source model: Sequence Diagram (*fly-to-target* operation)

profile, while the output model makes use of the HLA profile.

Specifically, the source model consists of by the following SysML diagrams:

- a set of BDDs and IBDs, that define the static structure of the system to be developed in terms of its physical components;
- a set of SDs, which specify the interactions among the system components.

The target model gives the following views, specified by use of UML diagrams:

- *Structural view*, constituted by a UML Class Diagram that shows the partitioning of the simulation model in terms of federates. The diagram also shows how federates publish or subscribe HLA resources (i.e., object classes and interaction classes);
- *Behavioral view*, constituted by a set of SDs that show the interactions among federates and RTI;

The model transformation has been specified in QVT language [13], which is provided by the Object Management Group (OMG) as the standard language for specifying model transformations that can be executed by available transformation engines [6, 10]. The following subsections specify the mapping rules defined to generate the structural view and the behavioral view, respectively[2].

[2]It should be noted that some stereotypes provided by the HLA profile (e.g., dimension, synchronization and switch) are used to represents FOM elements that cannot be automatically derived by the SysML model and, as such, are not considered by the transformation rules. The HLA profile has been designed in order to be as complete as possible, and suitable to fully describe an HLA federation, beyond the capabilities of the current prototypal implementation of the SysML-to-HLA transformation. In this respect, the HLA model could be manually refined.

Mapping rules for the structural view

The structural view of the target model, specified as a UML Class Diagram, is generated according to the following rules.

Rule 1: The main `block` element in the block definition diagram stereotyped as `<<federation>>` represents the HLA federation and is thus mapped to a UML `class` element, stereotyped as `<<federation>>`. Other block elements, stereotyped as `<<federate>>` in the BDD, represent the HLA federates that participate to the federation and are thus mapped to UML class elements stereotyped as `<<Federate>>`. Moreover, in the target model, such class elements representing federation and federates are connected by aggregation associations. Each `parameter` element associated to `block` elements stereotyped as `<<objectclass>` is mapped to a UML `class` element, stereotyped as `<<objectclass>>`.

Rule 2: A `block` stereotyped as `<<objectclass>>` and connected by a composition relationship with a block element representing a federate, is mapped to a UML `class` element, stereotyped as `<<objectclass>>`.

Rule 3: A `signal` element, stereotyped as `<<interactionclass>>`, is mapped to a UML `class` element, stereotyped as `<<interactionclass>>`. The related signal `properties` are mapped to class `attributes` in the target model, stereotyped as `<<interaction-parameters>>`.

Rule 4: Each composition association between a `block` A stereotyped as `<<federate>>` and `block` B stereotyped as `<<objectclass>>` is mapped to an association between the corresponding `class` elements C and D in the target model, stereotyped as `<<publish>>`. It is assumed that `class` elements C and D are generated from A and B by applying Rule 1 and Rule 2.

Rule 5: Each association between a `block` A stereoyped as `<<federate>>` and a `block` B stereotyped as `<<objectclass>>`, where a composition association exists between `block` B and a `block` C stereotyped as `<<federate>>`, is mapped to an association between the corresponding elements D and E in the target model, stereotyped as `<<subscribe>>`. `Class` elements D and E have been generated from A and B by applying rule 1 and rule 2.

Rule 6: Each association between `block` elements A and B, both stereotyped as `<<objectclass>>`, where i) `block` elements C and D are stereotyped as `<<federate>>` and ii) composition associations exist between A and C, and A and B, respectively, is mapped to an association between the corresponding elements E and F in the target model, stereotyped as `<<subscribe>>`. `Class` elements E and F have been generated from C and B by applying rule 1 and rule 2.

Rule 7: Each `association` that connects `flowports` belonging to `block` elements A and B, being C the exchanged `signal`, is mapped to two associations in the target model. The first one between the `class` element D and F, where D and F correspond to A and C according to rule 1, rule 2 and rule 3. The second one between the class element E and F, where E corresponds to B, according to rule 1 and rule 2. Such associations are stereotyped according to the flowport direction:

- `<<subscribe>>`, if the port `direction` is *out*;
- `<<publish>>`, if the port `direction` is *out*;
- both `<<publish>>` and `<<subscribe>>`, if the port `direction` is *inout*;

The rationale of mapping rules from 4 to 7, which are specifically involved in the generation of the publish/subscribe relationships, is depicted in Figure 5.

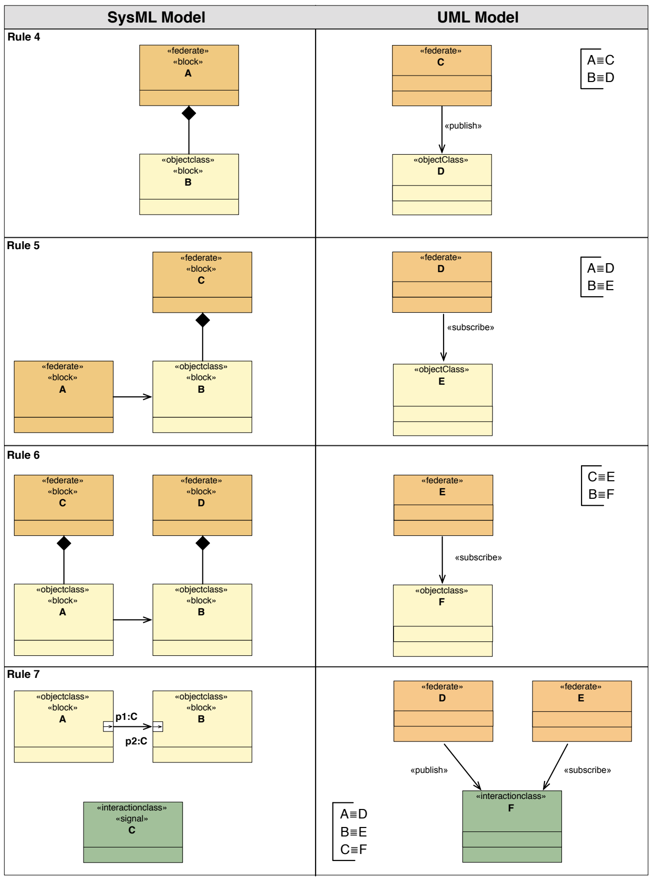

Figure 5. Mapping rules for *publish/subscribe* relationships

Mapping rules for the behavioral view

Sequence diagrams in the source model specify the interactions between model elements representing federates (i.e., `block` elements stereotyped as `<<federate>>`). The behavioral view of the target model is generated according to the following rules.

Rule 1: Each UML sequence diagram in the source model is mapped to a sequence diagram in the target model. Such diagrams constitutes the behavioral view of the target model.

Rule 2: In order to represent interactions between a federate and the HLA Run Time Infrastructure (RTI), each sequence diagram contains a UML component named `RTI`.

Rule 3: Each sequence diagram in the target model represents the behavior of a federate interacting with its components and/or other federates. The diagram must contains the following messages:

- a self message named *createRTIAmbassador()*, stereotyped as `<<initialization>>`;
- a message exchanged between federate and `RTI`, named *joinFederation()*, stereotyped as `<<initialization>>`;
- a set of messages echanged between federate and `RTI`, namely *publishInteractions()*, *subscribeInteractions()*, *publishObjects()* and *subscribeObjects()*, stereotyped as `<<initialization>>`, according to the publish/subscribe capability of each federate;
- main flow according to subsequent rules 4 and 5;
- a message exchanged between federate and `RTI`, named *leaveFederation()*, stereotyped as `<<message>>`;

Rule 4: messages included in the source sequence diagram exchanged between a federate and one of its component (i.e., an element stereotyped as `<<federate>>` and an element stereotyped as `<<objectclass>>`) are mapped to self messages to the federate, stereotyped as `<<action>>`.

Rule 5: messages included in the source sequence diagram exchanged between two federates are mapped to messages between federate and `RTI` (and vice-versa). Such messages are stereotyped as `<<messages>>`.

HLA-based UML simulation model of the AS System

Going back to the example application, Figures 6 and 7 depict the UML class diagram obtained from the SysML structural view and the UML sequence diagram obtained from the SysML behavioral view (i.e., the sequence diagram corresponding to the one shown in Figure 4), respectively. In this step existing federates suitable to be integrated in the federation are also identified.

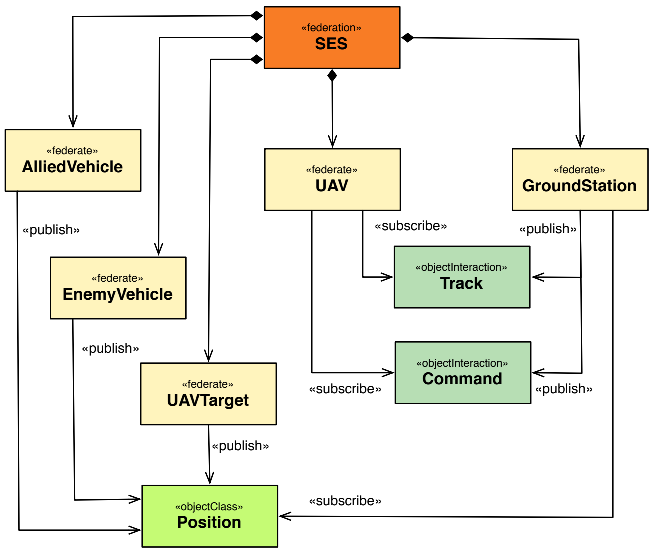

Figure 6. Target model: structural view

Federation Implementation

The last step of the method covers both the generation of the required simulation code and the integration of existing federates. In this respect, the `HLA-to-code` *model-to-text* transformation takes as input the HLA-based UML model produced in the previous step and yields as output the Java/HLA code that implements the HLA-based distributed simulation. It should be noted that the proposed method does not generate the complete code that implements the HLA-based distributed simulation. The model-driven method should be considered as a *supporting tool*. More specifically, the `HLA-to-code` transformation is executed to generate a template of the Java classes that contains the class structure (i.e., constructor, method and attribute declarations, exception management, etc.) and most of the HLA-related code (i.e., data types definition, RTI interaction methods, etc.), but the code implementing the federate simulation logic has to be added manually, as well as the code required to integrate existing federates.

Before discussing the so obtained AS executable simulation code, the `HLA-to-code` transformation is introduced in the next subsection.

HLA-to-Code Transformation

The `HLA-to-Code` model-to-text transformation takes as input the UML model of the HLA-based simulation and yields as output the corresponding implementation code, which is generated in Java and makes use of the *Pitch* HLA implementation [17].

The model-to-text transformation has been implemented by use of Acceleo [7], the model-driven Eclipse plugin for code generation. The implementation of the proposed transformation includes the following *templates*:

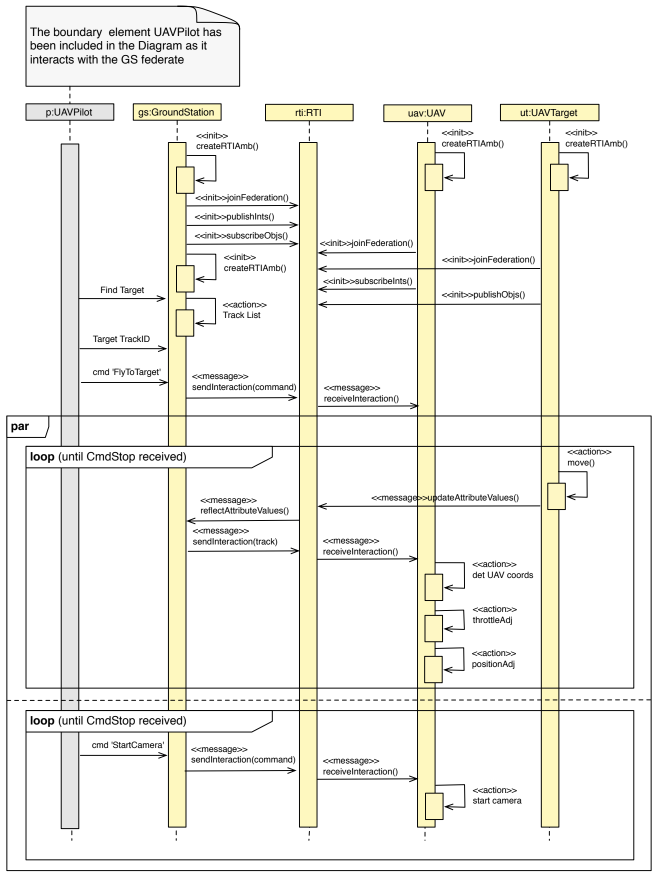

Figure 7. Target model: behavioral view (*fly-to-target* operation)

- *generateFederate:* for each element in the HLA model stereotyped as `<<federate>>`, this template generates a Java class that implements the corresponding federate;

- *generateObjIntClass:* for each UML class in the HLA model stereotyped as `<objectClass>>` or `<<interactionClass>>`, this template generates a tag in the XML file in which classes representing objectClass and interactionClass are serialized. In other words, such XML file specifies the FOM. Moreover, this template creates in each class generated by the *generateFederate* template the method for publishing and subscribing resources, according to the *Publish Subscribe relationship* in the structural HLA model.

- *generateAmbassador:* generates a set of Java classes constituting the implementation of the needed federate ambassadors.

- *completeFOM*: generates the FOM sections related to switch, synchronization points and dimension, while the corresponding elements are specified by the HLA model.

In this respect, the following code gives a portion of the resulting Java/HLA code corresponding to the implementation of the UAV federate:

```
package it.uniroma2.sel.simlab.ses.uav;
import hla.rti1516e.*;
import hla.rti1516e.exceptions.*;
```

```
...
public class UAV extends NullFederateAmbassador{
//create ambassador
RtiFactory rtiFactory = RtiFactoryFactory.getRtiFactory(); _ambassador =
       rtiFactory.getRtiAmbassador();
try {
_ambassador.connect(this, CallbackModel.HLA_IMMEDIATE,
localSettingsDesignator);
}catch (AlreadyConnected ignored)
...
_ambassador.joinFederationExecution(federateName + federateNameSuffix, "SES",
       federationName, new URL[] {url});
...
//defines handlers and publish/subscribe capabilities
try {
  getHandles();
  subscribeObjects();
} catch (FederateNotExecutionMember e) {
throw new RTIinternalError("HlaInterfaceFailure", e);
start();
}
```

CONCLUSION

This paper has introduced a model-driven method to support the code generation of an HLA-based simulation from a SysML specification of the autonomous system to be simulated. The approach makes use of two UML profiles for annotating both SysML and UML diagrams with HLA-based details and is founded on two model transformations that automatically map the source SysML model into an HLA-specific model and eventually into the Java/HLA source code. The proposed method enables system designers to obtain the executable HLA-based simulation code at a largely reduced effort and without being required to own specific DS skills, as shown by use of an example application dealing with the distributed simulation of an autonomous surveillance and engagement system.

REFERENCES

1. Antonova, I., and Batchkova, I. Development of Multi-Agent Control Systems using UML/SysML. In *Multi-Agent Systems - Modeling, Control, Programming, Simulations and Applications*, F. Alkhateeb, Ed. InTech, 2011.
2. Bocciarelli, P., D'Ambrogio, A., and Fabiani, G. A Model-driven Approach to Build HLA-based Distributed Simulations from SysML Models. In *Proceedings of the 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, SIMULTECH '12 (2012), 49–60.
3. Callow, G., Kalawsky, R., Watson, G., and Okuda, Y. Addressing systems verification of autonomous systems through Bi-directional model transformations: A systems model driven architecture approach. In *6th International Conference on System of Systems Engineering*, SoSE 2011 (2011), 311–316.
4. Castillo-Effen, M., Visnevski, N., and Subbu, R. Modeling and simulation for unmanned and autonomous system test and evaluation. In *Proceedings of the 9th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '09, ACM (New York, NY, USA, 2009), 86–92.
5. Cetinkaya, D., Verbraeck, A., and Seck, M. D. MDD4MS: A Model Driven Development Framework for Modeling and Simulation. In *Proceedings of the 2011 Summer Computer Simulation Conference*, SCSC '11, Society for Modeling & Simulation International (Vista, CA, 2011), 113–121.
6. Eclipse Foundation. *QVT Transformation Engine*. 2010. http://www.eclipse.org/m2m.
7. Eclipse Foundation. *Acceleo*. 2011. http://www.acceleo.org/pages/home/en.
8. Fujimoto, R. M. *Parallel and Distribution Simulation Systems*, 1st ed. John Wiley & Sons, Inc., New York, NY, USA, 1999.
9. Haouzi, H. E. Models simulation and interoperability using MDA and HLA. In *Proceedings of the IFAC/IFIP International conference on Interoperability for Enterprise Applications and Software (I-ESA'2006)* (2006).
10. IKV++ Technologies Ag. *Medini QVT*. 2008. http://projects.ikv.de/qvt.
11. Jimenez, P., Galan, S., and Gariia, D. Spanish HLA abstraction layer: towards a higher interoperability model for national. In *Proceedings of the European Simulation Interoperability Workshop* (2006).
12. OMG. *MDA Guide, v. 1.0.1*. 2003.
13. OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation, version 1.0*. 2008.
14. OMG. *MOF Model to Text Transformation Language (MOFM2T), 1.0*. 2008.
15. OMG. *System Modeling Language, v.1.2*. 2010. http://www.omg.org/spec/SysML/1.2/.
16. Parr, S., and Keith-Magee, R. The Next Step: Applying the Model Driven Architecture to HLA. In *Proceedings of the 2003 Spring Simulation Interoperability Workshop* (2003).
17. Pitch. *pRTIe*. 2012. http://www.pitch.se.
18. Tolk, A., and Muguira, J. A. M&S within the Model Driven Architecture. In *Proceedings of the Interservice/Industry Training, Simulation, and Education (I/ITSEC) Conference* (2004).