Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, eds.

MODEL-DRIVEN DISTRIBUTED SIMULATION ENGINEERING

Paolo Bocciarelli
Andrea D'Ambrogio
Andrea Giglio
Emiliano Paglia

Department of Enterprise Engineering
University of Rome Tor Vergata
Rome, Italy

ABSTRACT

Simulation-based analysis is widely recognized as an effective technique to support verification and validation of complex systems throughout their lifecycle. The inherently distributed nature of complex systems makes the use of distributed simulation approaches a natural fit. However, the development of a distributed simulation is by itself a challenging task in terms of effort and required know-how. This tutorial introduces an approach that applies highly automated model-driven engineering principles and standards to ease the development of distributed simulations. The proposed approach is framed around the development process defined by the DSEEP (Distributed Simulation Engineering and Execution Process) standard, as applied to distributed simulations based on the HLA (High Level Architecture), and is focused on a chain of automated model transformations. A case study is used in the tutorial to illustrate an example application of the proposed model-driven approach to the development of an HLA-based distributed simulation of a space system.

1 INTRODUCTION

The development of complex systems strongly benefits from the adoption of quantitative analysis techniques that enable an early evaluation of the system behavior, so to assess, before starting implementation or maintenance activities, whether or not the to-be system is going to satisfy the stakeholder requirements and constraints.

In this context, simulation-based techniques may be effectively introduced to enact the design-time evaluation of various structural and/or behavioral properties of the system under study. The adoption of simulation techniques is widely recognized as a cost-effective alternative to the development of experimental prototypes and as a valuable strategy to mitigate the risk of time/cost overrun due to redesign and re-engineering activities (Gianni et al. 2014).

The inherently compositional and distributed nature of complex systems makes the use of distributed simulation approaches a natural fit. A *distributed simulation (DS)* results from the orchestration of a number of simulation components that essentially mirror the component-based structure of the system under study.

However, the implementation of a DS is a challenging task in terms of the required effort and the significant know-how that is needed to properly use the available frameworks, such as the High Level Architecture (HLA), and the related implementation technologies (IEEE 2010b; IEEE 2010c; IEEE 2010d). In some cases, the development of a DS is seen as a task of complexity comparable to the development of the system under study (D'Ambrogio and Durak 2016).

For such reasons, the DS implementation activities are to be carried out according to a well-defined process, which is applied to provide a systematic and disciplined guide for DS developers, so to eventually

satisfy the DS requirements. This tutorial specifically addresses the definition of an *engineering process* that effectively supports the DS development through the use of formal models and the introduction of a significant degree of automation.

Systems development processes have traditionally focused on document-centric approaches based on the use of documents and data available at different level of abstractions and using different notations. A significant step forward has been achieved with the introduction of *model-based systems engineering (MBSE)* that, according to the definition promoted by the International Council on Systems Engineering (INCOSE), refers to the formalized application of modeling to support the system development, along all the different development phases (INCOSE 2017).

A relevant issue underneath the MBSE approach is the availability of a language that provides the required modeling capabilities. In this respect, SysML, a UML domain-specific extension for systems engineering applications, is currently considered as the standard modeling language adopted in the MBSE context (OMG 2017b). The advantages obtained by MBSE in terms of enhanced communications, reduced development risks, improved quality, increased productivity and enhanced knowledge transfer, can be further scaled up by advanced approaches that apply metamodeling techniques and automated model transformations, introduced in the more general *model-driven engineering (MDE)* context, to increase the level of automation throughout the system lifecycle.

In this respct, this tutorial introduces an approach that applies model-driven engineering principles and standards to reduce the effort of DS development. The proposed approach is framed around the development process defined by the DSEEP (Distributed Simulation Engineering and Execution Process) standard (IEEE 2010a), as applied to HLA-based distributed simulations (IEEE 2010b).

Specifically, the proposed approach, named *MoDSEEP (Model-driven DSEEP)*, introduces appropriate model-to-model and model-to-text transformations to automatically derive a significant portion of the HLA-based DS implementation from the SysML description of the system under study.

The rest of the paper is structured as follows. Section 2 gives the reader basic notions about model-driven engineering and HLA-based DS. Section 3 outlines MoDSEEP, the proposed DSEEP enhancement. Section 4 illustrates the SysML extension that is used to annotate SysML models given as input to the chain of model transformations outlined in Section 5. Section 6 describes an example application of the proposed approach and, finally, Section 7 gives concluding remarks.

2 BACKGROUND

The next two subsections provide the necessary background about the main concepts at the basis of this tutorial, i.e., model-driven engineering and HLA-based DS, respectively.

2.1 Model-driven Engineering

Model-driven engineering (MDE) is an approach to system design and implementation that addresses the raising complexity of execution platforms by focusing on the use of formal models (Atkinson and Kühne 2003; Schmidt 2006). According to this paradigm, the abstract models that are specified at the beginning of the system lifecycle are then given as input to *model transformations* that generate models at lower levels of abstraction, until stepwise refined models can be made executable.

One of the most important initiatives driven by MDE is the *Model Driven Architecture (MDA)*, the Object Management Group (OMG) incarnation of MDE principles (OMG 2003). The following main standards have been introduced as part of the MDA effort:

- *Meta Object Facility (MOF)*: for specifying technology neutral metamodels (i.e., models used to describe other models) (OMG 2017a);
- *XML Metadata Interchange (XMI)*: for serializing MOF metamodels/models into XML-based schemas/documents (OMG 2015);

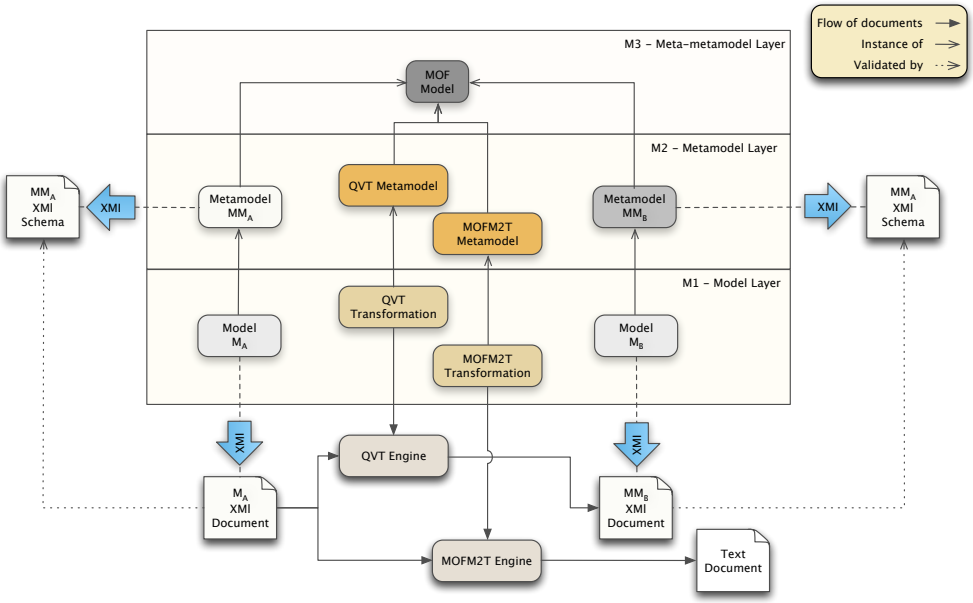

Figure 1: Overview of MDA standards.

- *Query/View/Transformation (QVT)* and *MOF Model To Text (MOFM2T)*: for specifying *model-to-model* and *model-to-text* transformations, respectively (OMG 2016; OMG 2008).

The relationship among the aforementioned MDA standards is summarized in Figure 1. Model M_A and model M_B are instances of their respective metamodels, namely metamodel MM_A and metamodel MM_B, which in turn are expressed in terms of *MOF Model* primitives. The *QVT metamodel*, which provides an hybrid (declarative/imperative) transformation language, is used to specify model-to-model transformations, e.g., to generate model M_B from model M_A. Both model M_A and model M_B can be serialized using XMI rules to obtain the corresponding XMI documents, that is, the XML-based representation of such models. XMI rules can also be used at metamodel layer to serialize metamodels and obtain XMI schemas for XMI document validation. When the target model is of text type (e.g., code written in a given programming language), the *MOFM2T metamodel* is used to specify the relevant model-to-text transformation.

2.2 Distributed Simulation and High Level Architecture

A DS results from the orchestration of loosely coupled simulation components that are executed on computational nodes interconnected through a network infrastructure of LAN or WAN type (Kuhl et al. 1999).

A relevant and widely adopted DS framework is the *High Level Architecture (HLA)*, initially developed by the US DoD Modeling and Simulation Coordination Office and then standardized by IEEE (IEEE 2010b; IEEE 2010c; IEEE 2010d). In essence, HLA provides a set of rules to enable the interoperability of distributed simulation components, according to the following main concepts:

- *Federation*: a DS execution composed of a set of simulation components.
- *Federate*: a simulation component that is part of a federation.
- *Run-time infrastructure (RTI)*: the simulation-oriented middleware for managing federates interaction.

A relevant issue covered by the HLA standard is the definition of the *Federation Object Model (FOM)*, which describes the data exchanged by federates during the federation execution. The FOM includes *object*

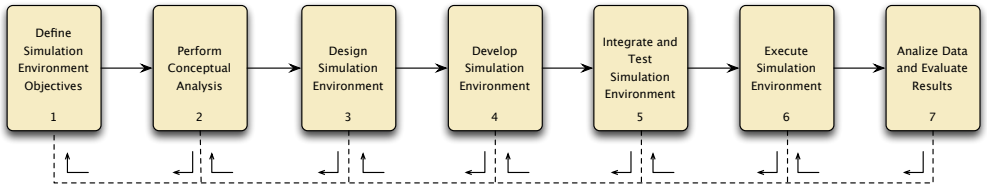

Figure 2: HLA Distributed Simulation Engineering and Execution Process (DSEEP).

classes and *interaction classes*. An object class is composed of a set of *object attributes* whose values define the state of a persistent object at any point during the DS execution, whereas an interaction class defines an event that a federate can generate or react to, as described by a set of event properties (or *interaction parameters*). The two types of information (objects and interactions) are exchanged through a publish/subscribe paradigm by using the services provided by the RTI. A federate can register an object, which is an instance of an object class, and then update attribute values. Other federates that are subscribed to that object class can discover the related instances and then receive attribute value updates. Interactions are used in a similar way, except that they have associated a set of parameters and do not represent persistent entities.

The HLA framework also exploits the Distributed Simulation Engineering and Execution Process (DSEEP) (IEEE 2010a), which defines a standardized and rigorous process for developing and executing distributed simulations. Figure 2 depicts the flow of steps that the DSEEP introduces to drive the development and the execution of a DS. The DSEEP underlines the importance of clearly identifying the simulation objectives and defining a conceptual model of the simulation. The design and the development of the federation are executed by allocating the required responsibilities to the various federates, and also by identifying those existing federates which can be reused and integrated in the implementation step.

3 MoDSEEP: MODEL-DRIVEN DISTRIBUTED SIMULATION DEVELOPMENT PROCESS

This section illustrates the proposed *MoDSEEP (Model-driven DSEEP*), an enhancement of DSEEP that benefits from the adoption of model-driven engineering principles and that has been tailored to fit the systems engineering domain's needs. The MoDSEEP rationale is shown in Figure 3. MoDSEEP has been specified as a flow of artifacts that are generated as the result of the following manual and automated (transformation-based) steps:

1. At the first step, the **Simulation Objectives and Constraints** are identified, in order to clarify what users expect from the simulation.
2. Then a **Conceptual Analysis** is carried out to identify the abstract concepts at the basis of the simulation specification. The conceptual analysis produces a SysML-based conceptual model and includes two sub-steps. First, the SysML4HLA profile, a SysML extension described in Section 4.1, is used to annotate the SysML model with specific marks used to derive the HLA-based simulation model. Specifically, the profile allows one to specify how the system has to be partitioned in terms of federates and how system model elements have to be mapped to HLA model elements, such as object classes and interaction classes. The second sub-step deals with the specification of the FOM Datatype Library. Such a library, that is introduced in section 5.3, specifies concepts and datatypes that are referred to the specific application domain and used for the automated generation of FOM modules.
3. The **Design Federation Environment** step is divided in two different sub-steps. First, the annotated system model produced at the previous step is given as input to the `SysML-to-HLA` *model-to-model* transformation to generate the HLA-based simulation model (Federation Model). In real world cases, a federation model may be composed of several interacting subsystems and may also include non-software components, as, e.g., for *hardware-in-the-loop* simulation. For such a reason,

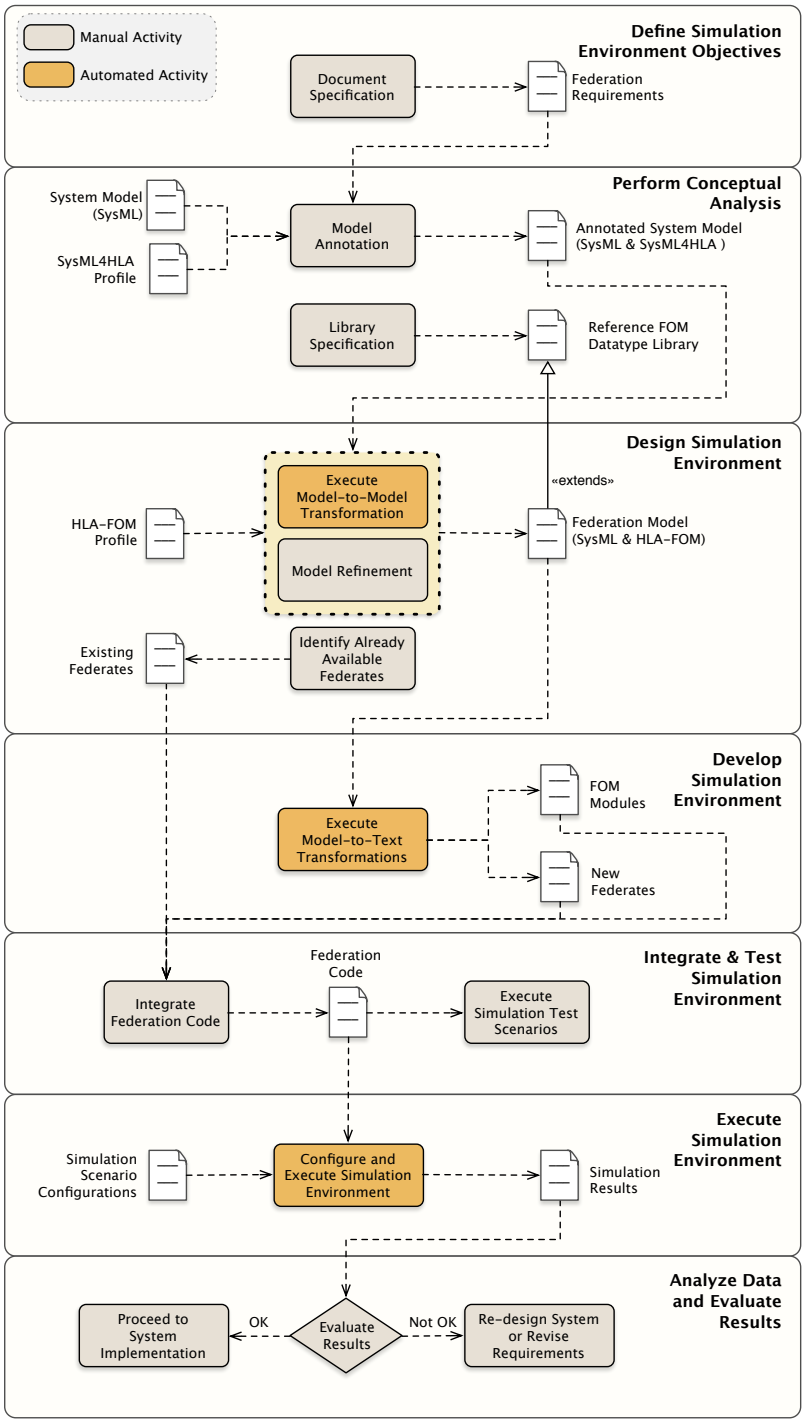

Figure 3: Outline of the MoDSEEP (Model-driven DSEEP).

the HLA-based simulation model is specified as a SysML model appropriately extended by use of the HLA-FOM profile, so to use concepts and datatypes specified by the FOM Datatype Library. The so obtained Federation Model allows one to specify the HLA role played by selected model elements (e.g., which SysML blocks act as federates, which block attributes have to be considered

as object parameters, etc.). The second sub-step deals with the possible identification of existing federates that are to be reused as part of the federation.

4. Once the federation model has been defined and the possible set of already available federates has been identified, the **Develop Federation Environment** step is carried out. Similarly to previous steps, the federation development consists of two sub-steps: the development of the required FOM modules and the implementation of federates to be developed from scratch. In this respect, MoDSEEP makes use of two *model-to-text* transformations: the `HLA-to-Code` transformation is used to generate a significant portion of the Java/HLA-based code of the federates, while the `HLA-to-FOM` transformation is in charge of generating the required set of FOM modules.
5. The **Integrate and Test Simulation Environment** step is then carried out to put all the pieces together. Existing federates identified at step 3 and new federates developed from scratch at step 4 are integrated to build the concrete implementation of the simulation system. A set of tests is also executed to discover and fix defects and to assess the correctness of the simulation behavior.
6. At the **Execute Simulation** step, the simulation experiments are finally run to investigate the behavior of the system under analysis.
7. Finally, the **Evaluation** step is used to analyze the simulation results so as to understand the behavior of the simulated system in its operational environment, and, ultimately, to assess whether or not a re-design is required before going ahead with the system implementation.

The tutorial specifically focuses on the *Conceptual Analysis*, *Design Federation Environment* and *Develop Federation Environment* steps of the MoDSEEP in Figure 3, as illustrated in the next two subsections, which give the details of the aforementioned SysML model annotations and the model transformations, respectively. The rest of steps are not directly impacted by the introduction of MoDSEEP, and thus are not further described.

4 SYSML MODEL ANNOTATIONS

As outlined in the previous section, MoDSEEP makes use of two SysML extensions, which have been introduced by use of a standard profiling technique, in other words a lightweight extension mechanism applied to a source modeling language to provide annotations that do not alter the original content of the model. In the MoDSEEP case, two profiles, namely *SysML4HLA* and *HLA-FOM*, have been introduced to annotate SysML models with HLA-related concepts at different levels of abstraction. Specifically, the main objective of the *SysML4HLA* profile is the identification of those SysML elements playing an active role in the HLA simulation (e.g., SysML blocks acting as object classes, or attributes to be considered as object parameters), while the *HLA-FOM* profile focuses on the representation of HLA-based implementation details (e.g., publish/subscribe relationships, time management, FOM modules, etc.). The two profiles, described in subsections 4.1 and 4.2, respectively, have been initially inspired by (Topçu et al. 2016).

4.1 SysML4HLA Profile

The *SysML4HLA* profile has been introduced to annotate SysML models with HLA-specific data, so as to drive the automated generation of the Federation Model used to implement the HLA-based DS of the system under study. *SysML4HLA* provides a set of stereotypes (or extensions of the source language elements) that extend the `Block` element of SysML, which in turn is an extension of the `Class` UML metaclass (or element of the UML metamodel). The introduced stereotypes are used to annotate those SysML `Block` elements that represent the basic elements of an HLA-based DS, i.e., federation, federates, object classes and interaction classes. Further details about the *SysML4HLA* profile can be found in (Bocciarelli et al. 2012; Bocciarelli et al. 2013).

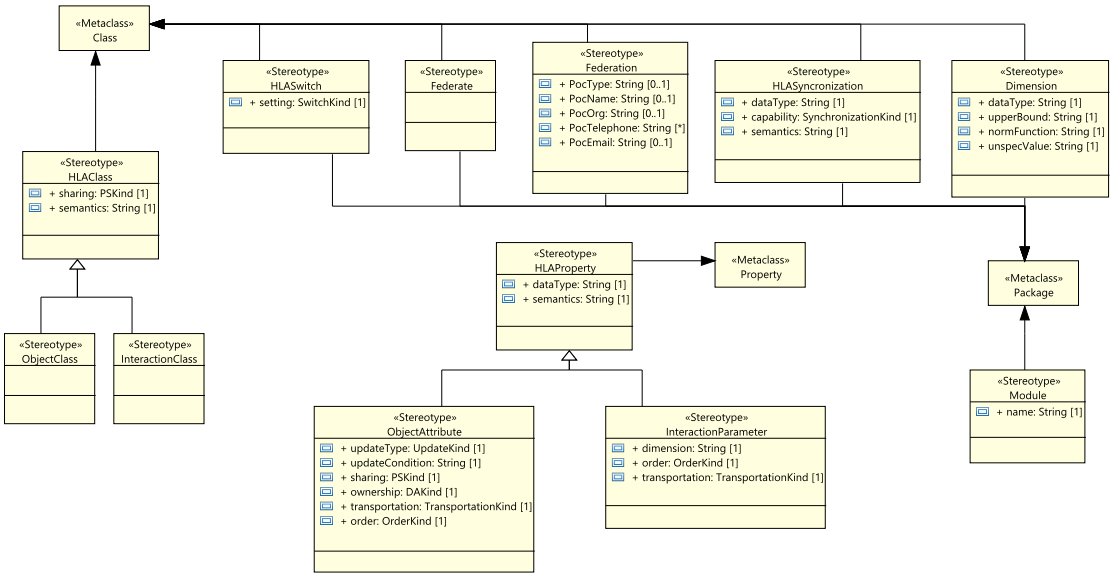

Figure 4: HLA-FOM SysML Profile: OMTKernel Package.

4.2 HLA-FOM Profile

The *HLA-FOM* Profile consists of a set of stereotypes organized in two different packages: the *OMTKernel* package, which contains stereotypes used to annotate the model, and the *HLADatatypes* package, which provides the data types of the attributes used to specify the stereotypes included in the *OMTKernel* package. Both packages refer to the HLA *Object Model Template (OMT)*, the standard that defines the format of HLA FOMs. Figure 4 illustrates the *OMTKernel* package. The reader is sent to (IEEE 2010d) for a detailed description of OMT types and objects, and to (Bocciarelli et al. 2019) for the full specification of the HLA-FOM Profile.

5 MODEL TRANSFORMATIONS RATIONALE

This section outlines the rationale of the following model transformations, which are at the basis of MoDSEEP:

- the `SysML-to-HLA` *model-to-model* transformation, which takes as input the SysML-based system model and yields as output the HLA-based federation model;
- the `HLA-to-FOM` *model-to-text* transformation, which takes as input the HLA-based federation model and yields as output the required set of FOM modules;
- the `HLA-to-Code` *model-to-text* transformation, which takes as input the HLA-based federation model and yields as output the code that partially implements the federates of the HLA-based DS.

5.1 Generation of Federation Model: SysML-to-HLA model-to-model transformation

The `SysML-to-HLA` model-to-model transformation is specified in the QVT Operational Mapping (QVTo) langauge (OMG 2016) and executed by use of the QVTo implementation provided as a plugin of the Eclipse Modeling Framework (EMF) (Eclipse Foundation 2008). The transformation takes as input a SysML model representing the system under study and yields as output a SysML model that specifies the HLA-based DS. The input model is annotated with stereotypes provided by the SysML4HLA profile, whereas the output model makes use of the HLA-FOM profile. The rationale of the transformation is described in Figure 5, which outlines how system model elements are mapped to federation model objects. For the

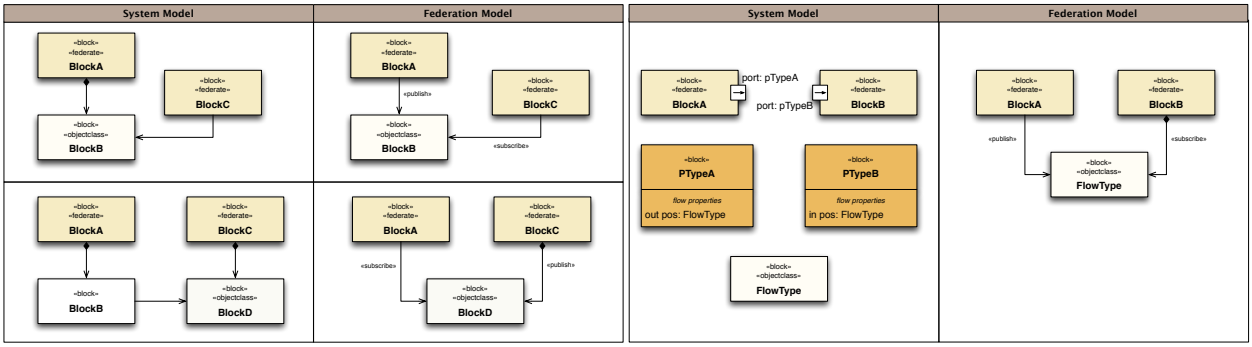

Figure 5: Rationale of the SysML-to-HLA model transformation.

sake of conciseness, the figure only shows source model elements stereotyped as `<<ObjectClass>>`. The transformation of elements stereotyped as `<<InteractionClass>>` follows an identical approach. The complete specification of the `SysML-to-HLA` can be found in (Bocciarelli et al. 2012; Bocciarelli et al. 2019; Bocciarelli et al. 2013).

5.2 Generation of HLA Code: HLA-to-Code model-to-text transformation

The `HLA-to-Code` model-to-text transformation takes as input the HLA-based federation model (i.e., the SysML-based Federation Model) and yields as output the corresponding HLA-based implementation code. The generated code makes use of the *Pitch pRTI* RTI implementation (Pitch 2012) and Java as the language for implementing federates and ambassadors. The model-to-text transformation is specified in the template-based Acceleo language (Eclipse Foundation 2012), which conforms to the OMG MOFM2T standard, and executed by use of the Acceleo implementation provided as an EMF plugin. The specification of the proposed transformation includes the following templates:

- *generateFederate*, which generates a Java class that implements the federate skeleton for each SysML block stereotyped as `<<Federate>>`, .
- *generateAmbassador*, which generates a set of Java classes implementing federate ambassadors.
- *generateObjIntClass*, which creates the skeleton of methods for publishing and subscribing resources for each SysML block stereotyped as `<<ObjectClass>>` or `<<InteractionClass>>`, according to Publish/Subscribe relationships and for each relevant class generated by the *generateFederate* template.

5.3 Generation of FOM Modules: HLA-to-FOM model-to-text transformation

The `HLA-to-FOM` model-to-text transformation makes use of the HLA-FOM profile, which allows one to specify implementation-oriented HLA-based annotations, and, optionally, a library specifying the datatypes and the objects hierarchy which have to be used in order to make the simulation compliant to a given reference FOM. Table 1 outlines the rationale of the model-to-text transformation by identifying the source of (where to find) the various input elements that the transformation uses to generate the corresponding FOM module elements. Beyond the mappings shown in Table 1, a complete understanding of the `HLA-to-FOM` transformation behavior requires the illustration of two fundamental issues: the subclassing of FOM entities and the partitioning of the FOM into different modules, as described in Section 5.3.1 and Section 5.3.2, respectively.

5.3.1 Handling of Subclassing

An important objective of the proposed approach is to specify a model transformation that is general enough to be used in different application domains and operational contexts. In other words, the transformation

Table 1: Data sources for FOM module elements.

Element	Source
Identification Table	
Name, Type	SysML model
Modification date	date of generation
POC	Profile
Object Classes	
Object Class name	SysML model
Object Class semantics	Profile
Class Hierarchy	SysML model + library
Object Attributes	
Attribute name	SysML model
Attribute datatype	SysMl model + library
Attribute semantics	Profile
Attribute characterization	Profile
Synchronization points	
All attributes	Profile

Element	Source
Interaction Classes	
Interaction Class name	SysML model
Interaction Class semantics	Profile
Class Hierarchy	SysML model + library
Interaction Parameters	
Parameter name	SysML model
Parameter datatype	SysMl model + library
Parameter semantics	Profile
Parameter characterization	Profile
Datatypes	
Name, type, hierarchy, semantics	SysML library
Switches	
All attributes	Profile

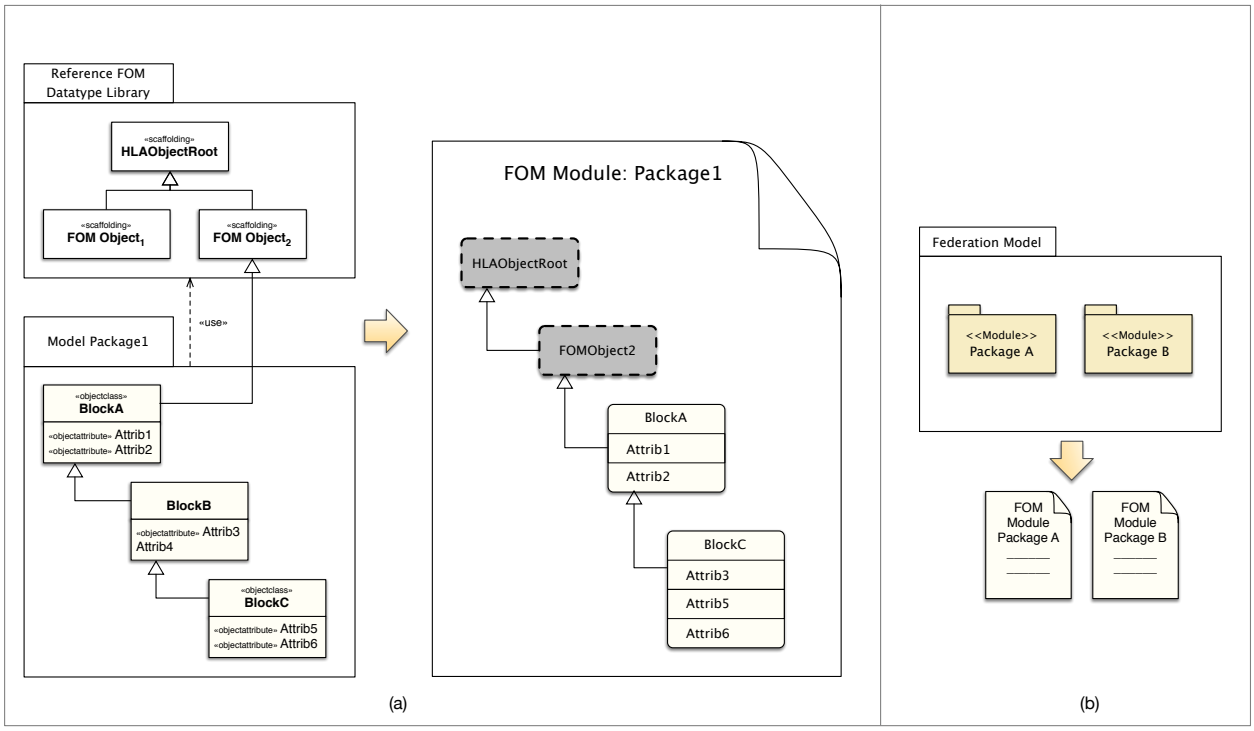

Figure 6: Handling of subclassing (a) and modularity (b).

must not assume the availability of a specific reference FOM and must not hard-code any predefined FOM structure. To achieve such an objective, the proposed approach makes use of an external SysML library, which defines the datatypes and the objects hierarchy as specified by the optional reference FOM.

In this respect, the availability of the external (and optional) library allows systems engineers to make the model compliant to the reference FOM by specifying that i) a model element stereotyped as `<<ObjectClass>>` or `<<InteractionClass>>` is a subclass of a given library element and, ii) an attribute of the source model stereotyped as `<<ObjectAttribute>>` or `<<InteractionParameter>>` must be typed by one of the datatypes provided by the library.

This design choice requires the `HLA-to-FOM` model-to-text transformation to effectively handle inheritance relationships by considering two different types of inheritance relationship. To better explain such an idea, let us consider the example shown in part (a) of Figure 6, where a SysML model uses an external Datatype Library.

The left part shows the model given as input to the transformation engine. The Datatype Library provides an implementation of the entity hierarchy specified in the Reference FOM and the implementation of the several related datatypes, as well.

The first type of inheritance relationship that the transformation has to address is the one existing in the actual annotated SysML model: as shown in the figure, `BlockC` is a subclass of `BlockB`, which in turn is a subclass of `BlockA`. The stereotypes `<<ObjectClass>>` and `<<ObjectAttribute>>` are used to specify which elements play the relevant role in the HLA federation (it is not necessary that all the elements in the model have a counterpart in the FOM).

The second type of inheritance relationship that the transformation has to address is the one between model elements and the elements of the imported library, when available, e.g., the subclassing with respect to the entities provided by the Reference FOM. In the example shown in part (a) of Figure 6, `BlockA` plays the role of an object class that inherits from the $FOM\ Object_2$ element, so to properly manage the scaffolding mechanism that makes the resulting FOM module compliant to the Reference FOM.

5.3.2 Handling of Modularity

Another relevant requirement that the model transformation has to satisfy is the handling of modularity. From a general perspective, the SysML model provides a representation of the entire DS. According to the HLA standard, the FOM is not necessarily to be described in a monolithic document, but can be specified throughout a set of modules, each providing a subset of the whole model. In this respect, the proposed method enables the partitioning of the FOM into a set of modules using the following strategy:

- the SysML system model is partitioned into different packages, so that model elements that have to be mapped to the same FOM module are contained in the same package;
- each package is annotated with the `<<Module>>` stereotype provided by the HLA-FOM profile, with the tag `Name` used to specify the related module name;
- the model transformation processes one package at a time, in order to yield as output the required FOM modules, as shown in part (b) of Figure 6.

6 EXAMPLE APPLICATION: DEVELOPMENT OF A SPACE SYSTEM FEDERATION

This section illustrates an example application of the proposed MoDSEEP. The provided example focuses on the development of a space system and illustrates the various steps that allows systems engineers to generate the HLA federation, starting from an abstract SysML-based specification of the system. The objective of this section is to show in more detail the MoDSEEP steps and the related *model-to-model* and *model-to-text* transformations, rather than focusing on the modeling of a full-fledged system or on the evaluation of the simulation results.

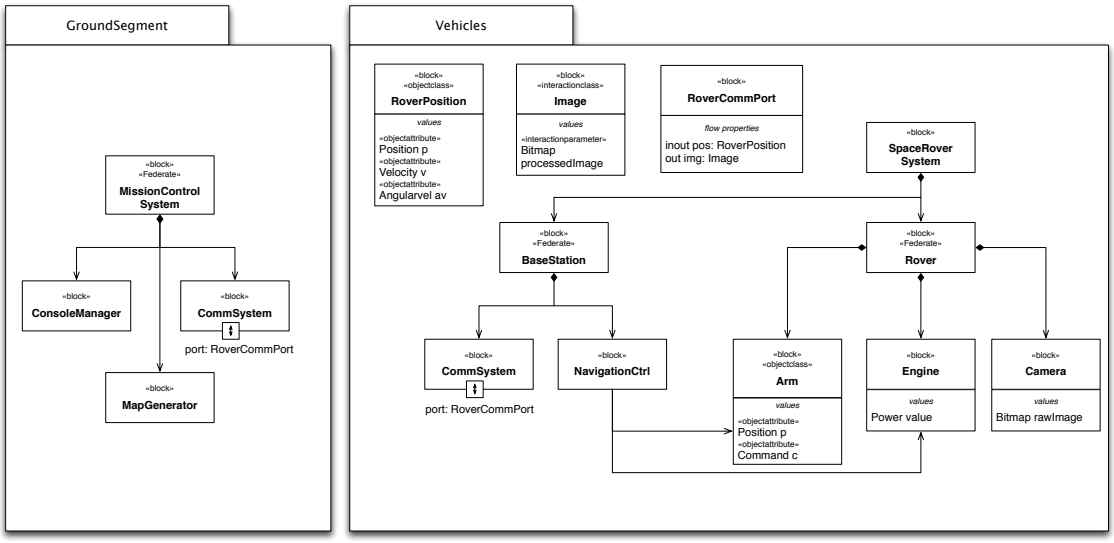

Figure 7: SysML model for the example space system, annotated with the SysML4HLA profile.

6.1 System Specification and Definition of the Simulation Requirements

Let us consider a space system dealing with the remote control of a space rover that moves on a planet surface. The space system is structured in two different subsystems: the *Space Rover* subsystem, which is turn composed by a base station and a vehicle equipped with a camera and a robotic arm, and the *Ground Segment*, which implements the subsystem to remotely control the rover from the Earth.

The definition of the simulation objectives and requirements allows systems engineers to identify and specify what users expect from the simulation, and to describe the operational scenarios used to execute the simulation. In this respect, let us suppose that the simulation must assess the behavior of the communication system that interconnects the ground station and the space rover, and also verify how safely the rover can be moved on the planet surface. As the simulation deals with the space domain, a relevant requirement that must be addressed to ensure interoperability in the given context, is that the federation must be compliant with the SISO Space Reference FOM (Möller et al. 2016). Finally, the federation must use a terrain representation that will be used to effectively simulate the Rover movements on the planet surface.

6.2 Perform the Conceptual Analysis: annotation of the System Model

The conceptual analysis step includes two different activities: the annotation of the system model with stereotypes provided by the SysML4HLA profile and the specification of the datatype library related to the adopted reference FOM. Figure 7 shows the annotated SysML model for the example space system.

6.3 Perform the Federation Design: generation of the Simulation Model

The design of the federation environment consists of two activities. The core of the federation design step is the execution of the `SysML-to-HLA` model-to-model transformation, which takes as input the annotated system model specified at the previous step and yields as output the model of the HLA federation. Such a model is annotated with stereotypes provided by the *HLA-FOM* profile, to specify the HLA-related information required to drive the additional transformation that generates the HLA implementation code.

Moreover, the federation model makes use of the datatype library that implements concepts and datatypes specified in the adopted reference FOM. It is worth noting that the use of a model-driven approach that supports the automated model generation may be supplemented by a manual activity carried out both to refine the federation model and to add modeling elements that do not exist in the system domain.

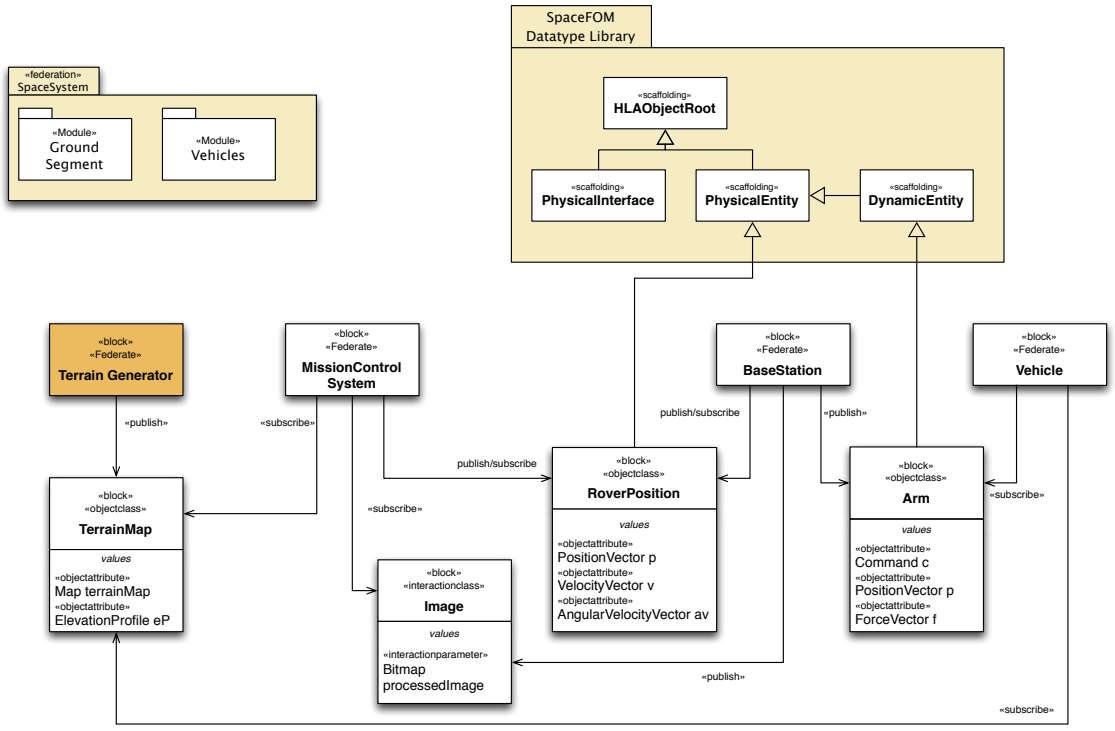

Figure 8: Space System Federation Model.

The so-obtained federation model for the example application is illustrated in Figure 8. The package diagram shown in the upper-left corner specifies that two different FOM modules are to be generated: one for the ground segment, and one for the vehicle. Moreover, since the federation makes use of the Space Reference FOM, the `RoverPosition` object class is a subclass of the `PhysicalEntity` class, while the `Arm` object class is a subclass of the `DynamicEntity` class. Finally, the `TerrainGenerator` federate is an example of a model element added during the manual refinement.

The model transformation generates HLA model elements (e.g., federates, object classes, etc.) that have a counterpart in the system model. In many real world cases, the federation model may include components that exist only in the federation domain. For the sake of conciseness, additional details such as datatypes included in the datatype library and stereotype tagged values have been omitted in Figure 8.

The second activity included in the Federation Design step deals with the identification of existing federates which can be used in the federation implementation. In the example application case, it is assumed that an already existing implementation of the *Terrain Generator* federate is adapted and integrated in the addressed federation.

6.4 Develop Federation Environment

The Develop Federation Environment step consists of the execution of two different *model-to-text* transformations. The first one, namely the `HLA-to-Code` transformation, generates the Java/HLA code implementing a significant portion of the required federates. Specifically, the transformation generates the skeleton of the Java classes (i.e., constructors, methods and attributes declarations, exception management) and most of the HLA-related code (i.e., data types definition, RTI interaction methods). The code implementing each federate simulation logic has to be added manually. As an example, a portion of the code that implements the *Vehicle* federate is shown in Listing 1.

The second transformation, namely the `HLA-to-FOM` transformation, takes as input the federation model and the datatype library and yields as output the complete set of required FOM modules. According

Listing 1: Fragment of Vehicle federate code.

```
package <package full name should be placed here>
import hla.rti1516e.*;
import hla.rti1516e.exceptions.*;
...
public class Vehicle extends FederateAmbassador{
...
//create ambassador
RtiFactory rtiFactory = RtiFactoryFactory.getRtiFactory(); _ambassador = rtiFactory.
    getRtiAmbassador();
try {
   _ambassador.connect(this, CallbackModel. HLA_IMMEDIATE, localSettingsDesignator);
}catch (AlreadyConnected ignored)
...
//join federation _ambassador.joinFederationExecution(federateName +
    federateNameSuffix, "MapViewer", federationName, new URL[] {url});
...
//defines handlers and publish/subscribe capabilities
try{ getHandles();
   subscribeObjects();
   publishInteractions();
} catch (FederateNotExecutionMember e){
throw new RTIinternalError ("HlaInterfaceFailure", e);
...
//start() method must provide the federate simulation logic;
...
}
```

to the annotation specified in Figure 8, the transformation output includes the *GroundSegment* and the *Vehicles* modules.

A fragment of the *Vehicles* FOM module is shown in Listing 3, while Listing 2 includes a fragment of the Space Reference FOM. The object classes of the *Vehicles* FOM conform to the hierarchy specified by the Space Reference FOM via the scaffolding declaration mechanism.

According to the remaining steps included in the MoDSEEP, once the several federates composing the space system federation and the required FOM modules are available, the distributed simulation is finally executed to assess the behavior of the space system in the addressed operational scenarios, and thus either proceed to system implementation activities or carry out system redesign or requirements revision activities.

7 CONCLUSIONS

This tutorial paper has introduced an approach that applies model-driven engineering principles to the cost-effective development of distributed simulations. The proposed approach, named MoDSEEP (Model-driven DSEEP), is framed around the development process defined by the DSEEP standard, as applied to HLA-based distributed simulations, and aims at enabling systems engineers to reduce the effort required to carry out HLA-based system analysis activities.

The tutorial is intended to illustrate how standards and technologies introduced in the model-driven engineering field can be effectively used to enhance the existing DSEEP standard process and overcome the main difficulties that often limit the adoption of HLA-based DS approaches.

The proposed MoDSEEP is founded on the execution of automated model transformations that allow systems engineers to iteratively refine an abstract SysML-based specification of a given system, so to obtain first an intermediate model of the related HLA-based federation and then both the skeleton of the Java-HLA DS implementation and the required set of FOM modules that specify the data exchanged by federates.

An integrated toolchain has been set up to implement both the profiles and the model transformations introduced by the MoDSEEP. The toolchain makes use of the various tools integrated into the Eclipse platform and the EMF project.

An example application to the development of an HLA-based DS for a space system has been used to illustrate the MoDSEEP benefits, as well as its limitations, when applied to the generation of an HLA implementation that conforms to a specific reference FOM.

Listing 2: Fragment of Space Reference FOM.

```
<?xml version="1.0" encoding="UTF-8"
    standalone="yes"?>
<objectModel xsi:schemaLocation="http://
    standards.ieee.org/IEEE1516-2010"
...
<dataTypes>
<simpleData>
  <name>Velocity</name>
  <representation>HLAfloat64LE
  </representation>
...
<arrayData>
  <name>VelocityVector</name>
  <dataType>Velocity</dataType>
  <cardinality>3</cardinality>
  <encoding>HLAfixedArray
  </encoding>
...
</arrayData>
</dataTypes>
<objectClass>
  <name>HLAobjectRoot</name>
  <objectClass>
  <name>PhysicalEntity</name>
  <sharing>PublishSubscribe
  </sharing>
  <semantics>A PhysicalEntity is...</
      semantics>
...
</objectModel>
```

Listing 3: Fragment of Vehicles FOM Module.

```
<?xml version="1.0" encoding="UTF-8"
    standalone="yes"?>
<objectModel>
<modelIdentification>
  <name>Vechicles Module</name>
        <type>FOM</type>
        <version>0.1</version>
...
</modelIdentification>
...
<objects>
<objectClass>
  <name>HLAobjectRoot</name>
  <objectClass>
    <name>PhysicalEntity</name>
    <objectClass>
      <name>RoverPositiom</name>
      <sharing>PublishSubscribe</sharing>
      <attribute>
        <name>v</name>
        <dataType>VelocityVector
        </dataType>
        <updateType>Conditional
        </updateType>
        <sharing>PublishSubscribe
        </sharing>
        <transportation>HLAreliable
        </transportation>
  ...
</objectModel>
```

REFERENCES

Atkinson, C., and T. Kühne. 2003. "Model-driven development: a metamodeling foundation". *Software, IEEE* 20(5):36–41.

Bocciarelli, P., A. D'Ambrogio, and G. Fabiani. 2012. "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, 49–60.

Bocciarelli, P., A. D'Ambrogio, A. Falcone, A. Garro, and A. Giglio. 2019. "A model-driven approach to enable the simulation of complex systems on distributed architectures". *SIMULATION*:1–27.

Bocciarelli, P., A. D'Ambrogio, A. Giglio, and D. Gianni. 2013. "A SaaS-based automated framework to build and execute distributed simulations from SysML models". In *Winter Simulations Conference: Simulation Making Decisions in a Complex World, WSC 2013, Washington, DC, USA, December 8-11, 2013*, edited by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. Kuhl, 1371–1382: IEEE.

Bocciarelli, P., A. D'Ambrogio, A. Giglio, and E. Paglia. 2019. "Automated Generation of FOM Modules for HLA-based Distributed Simulations". In *Proceedings of the 2019 SCS Spring Simulation Conference*. Tucson, AZ, USA.

D'Ambrogio, A., and U. Durak. 2016. "Setting systems and simulation life cycle processes side by side". In *2016 IEEE International Symposium on Systems Engineering (ISSE)*, 1–7.

Eclipse Foundation 2008. "Eclipse Modeling Framework Project (EMF)". http://www.eclipse.org/modeling/emf, last accessed on 26/6/19.

Eclipse Foundation 2012. "Acceleo". https://www.eclipse.org/acceleo, last accessed on 26/6/19.

Gianni, D., A. D'Ambrogio, and A. Tolk. (Eds.) 2014. *Modeling and Simulation-Based Systems Engineering Handbook*. CRC Press.

IEEE 2010a. "1730 - Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP)".

IEEE 2010b. "IEEE 1516 - Standard for Modeling and Simulation High Level Architecture - Framework and Rules".

IEEE 2010c. "IEEE 1516.1 - Standard for Modeling and Simulation High Level Architecture - Federate Interface Specification".

IEEE 2010d. "IEEE 1516.2- Standard for Modeling and Simulation High Level Architecture - Object Model Template (OMT) Specification".

INCOSE 2017. "International Council on Systems Engineering (INCOSE). Systems Engineering Vision 2020, Version 2.03 - INCOSE Document Number INCOSETP-2004-004-02".

Kuhl, F., R. Weatherly, and J. Dahmann. 1999. *Creating computer simulation systems: an introduction to the high level architecture*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Möller, B., E. Z. Crues, D. Dexter, A. Garro, A. Skuratovskiy, and A. Vankov. 2016. "A First Look at the Upcoming SISO Space Reference FOM". In *SISO 2016 Simulation Innovation Workshop (SIW)*.

OMG 2003. *MDA Guide revision 2.0 (ormsc/14-06-01)*.

OMG 2008. *MOF Model to Text Transformation Language (MOFM2T), 1.0*.

OMG 2015. *XMI - XML Metadata Interchange, version 2.5.1*.

OMG 2016. *Meta Object Facility (MOF) Query/View/Transformation, version 1.3*.

OMG 2017a. *Meta Object Facility (MOF) Specification, version 2.4.2*.

OMG 2017b. "System Modeling Language, v.1.5". https://www.omg.org/spec/SysML/1.5/, last accessed on 26/6/19.

Pitch 2012. "pRTIe". http://www.pitch.se/, last accessed on 26/6/19.

Schmidt, D. C. 2006. "Model-Driven Engineering". *IEEE Computer* 39(2).

Topçu, O., U. Durak, H. Oğuztüzün, and L. Yilmaz. 2016. *Distributed Simulation: A Model Driven Engineering Approach*. Springer.

AUTHOR BIOGRAPHIES

PAOLO BOCCIARELLI is a postdoc researcher at the Department of Enterprise Engineering of the University of Roma "Tor Vergata" (Italy). His research interests include software and systems engineering, business process management, model-driven development and distributed simulation. His email address is paolo.bocciarelli@uniroma2.it.

ANDREA D'AMBROGIO is associate professor of systems and software engineering at the Department of Enterprise Engineering of the University of Roma "Tor Vergata" (Italy). His research interests are in the areas of system performance and dependability engineering, model-driven systems and software engineering, business process management, and distributed simulation. His email address is dambro@uniroma2.it.

ANDREA GIGLIO is a postdoc researcher at the Enterprise Engineering Department of the University of Rome "Tor Vergata" (Italy). His research interests include model-driven system and software engineering and business process management. His email address is andrea.giglio@uniroma2.it.

EMILIANO PAGLIA is postdoc researcher at the Enterprise Engineering Department of the University of Rome "Tor Vergata" (Italy). His research interests include model-driven engineering and business process modeling and analysis. His email address is emiliano.paglia@uniroma2.it.