Developing HLA distributed simulations using Model-Driven DSEEP with OPM

João Gabriel da Cunha Schittler[1], **Alexandre Chagas Brites**[1], **Henrique de Oliveira Gressler**[1], **Raul Ceretta Nunes**[1]

[1]Graduate Program in Computer Science – Technological Center
Federal University of Santa Maria (UFSM) – Santa Maria – RS – Brazil

{jgschittler,acbrites,ceretta}@inf.ufsm.br

henriquegressler@gmail.com

Abstract. *Distributed simulations aim to replicate real-world behavior via computer networks. Developing source code for such simulations, especially while following standards like High-Level Architecture (HLA), is often error-prone due to its complexity, prompting code generators requiring abstract models. This work proposes a model-driven methodology that explores the Object-Process Methodology (OPM) modeling language in conjunction with UML to generate accurate and readable high-level models in DSEEP and transform them into code. For results, we present a case study in which a stakeholder with no prior knowledge of OPM was tasked and succeeded with modeling a distributed simulation to show how learnable and understandable OPM is, bridging the gap between conceptualization and implementation.*
Keywords *Distributed Simulation, High-Level Architecture, Model Driven Architecture, Object Process Methodology.*

1. Introduction

In the industries that produce distributed simulations (DS), interoperability among serious games (simulators) is an inherent requirement. One of the most prevalent standards for achieving this interoperability is the High-Level Architecture (HLA) [IEEE 2010], which leverages the publish/subscribe architecture as its foundational framework. Within the HLA framework, a DS is referred to as a "Federation," and each interconnected simulator assumes the role of a "Federate" within that federation. HLA also defines the utilization of a Federation Object Model (FOM) file to delineate the objects, interactions, and data types that will be exchanged over the federation network. The widespread adoption of HLA transcends various sectors, encompassing domains such as industry [Possik et al. 2023], gaming [Choi et al. 2013], healthcare [Petty e Windyga 1999], military [Lee et al. 2005], and space [Crues et al. 2022]. However, there are two widely known challenges when developing HLA distributed simulations [Möller et al. 2006] [Graham 2007]: i) to design an adequate and concise FOM file that aligns with the simulation's objectives; and ii) to accurately implement the simulation source code to conform to the established and ensure effective execution of the simulation environment.

In game development, HLA can provide a common interface from which different games can offer interoperability. This is particularly useful for simulator-type games. It can also be used as a solution for distributed server architectures, helping overcome scalability problems [Lees et al. 2006].

The Distributed Simulation Engineering and Execution Process (DSEEP) [IEEE 2022] is a widely recognized development process for distributed simulations that helps HLA-based developing processes. DSEEP defines clear steps to be followed to create a distributed simulation, from the definition of the simulation objectives and the conceptual modeling to the development and testing of the simulation.

Despite DSEEP, developing HLA simulations comes with its own set of challenges. Implementing the simulation source code can be quite error-prone and complex, requiring at least the use of automatic code generators from the federation object model (FOM) [dos Santos e Nunes 2022]. Model-Driven Architecture[1] (MDA) is an approach to software design that provides guidelines for structuring software specifications expressed as models, separating business and application logic from underlying platform technology. In MDA, the simulation conceptual model can be, for example, a set of SysML[2] diagrams [Bocciarelli et al. 2019a], but the effort to produce the required output to solve the requirements can often be a complex task [Mall 2018]. It happens mainly when the separation of business and application logic is unclear.

Assuming the main problem is to coordinate the stakeholders' and project developers' understanding, in this paper, we propose a new methodology for developing HLA distributed simulations. The methodology implements a Model-Driven DSEEP exploring Object-Process Methodology (OPM) [Dori 2002] and Unified Modeling Language (UML). The use of MDA to add automatic model transformations on DSEEP was first explored in Bocciarelli [Bocciarelli et al. 2019b] and D'Ambrogio [D'Ambrogio et al. 2019], where SysML was explored as a modeling language. However, SysML may not be the most appropriate language for the first conceptual model due to having multiple diagrams to properly represent the system's structure and behavior. There are many known challenges in user comprehension and complexity associated with using multiple diagram types [Ong e Jabbari 2019]. Furthermore, the high information density required to fill the many diagrams that compose a SysML model may not be well defined at the initial modeling [Šenkỳr e Kroha 2021]. So, using conceptual modeling language that requires less information to describe the simulation and is more readable can be of greater utility in the first steps of software development approaches [Basnet et al. 2020].

In our proposed methodology, after the acquisition of the simulation´s requirements, an OPM model is developed, containing complete diagrams for the abstract views of the system. Depending on the amount of information described in the requirements, more detailed diagrams also be made for less abstract parts of the system. The stakeholders review this OPM model, and the missing information is filled in. The finished OPM model is then transformed into a UML class diagram that is automatically transformed until it becomes the simulation's FOM and HLA source code.

In summary, the main contributions of this work are i) A new industry approach to build HLA distributed simulation conceptual modeling that is more readable for people with less knowledge of modeling languages. With greater model readability, the readers will better understand the simulation's conceptual vision, increasing the alignment between what the stakeholders want the project to be and what the developers interpret. Thus, this contribution enables easier conceptual modeling and reduces the risks of

[1]MDA specification available at https://www.omg.org/mda/

[2]SysML language specification available at https://www.omg.org/spec/SysML/

changes in project requirements; and ii) An implementation framework of such modeling step inside a complete HLA simulation development methodology based on DSEEP and using OPM and UML as the modeling languages, QVT-Operacional[3] as the language used for model-to-model transformations, and the StringTemplates template engine for code generation.

The rest of this work is organized as follows: Section 2 presents the concepts required to understand this work's proposition; Section 3 provides the explanation of this work's proposed methodology; Section 4 briefly explains how the methodology was implemented; Section 5 presents a case study that shows how someone with no previous knowledge of OPM learned and modeled a DS; Section 6 discusses and compares related works; Finally, Section 8 provides this work's conclusions.

2. Background

2.1. HLA and DSEEP

When developing HLA federations that will run on a middleware called Run Time Infrastructure (RTI), the FOM design must be aligned with the simulation's goals, and the simulation code must be correct and efficient. To facilitate this, the creation of conceptual models and the use of model transformations until automatic code generation has been explored in HLA-based simulation development.

DSEEP has seven main steps: Define Simulation Environment Objectives; Perform Conceptual Analysis; Design Simulation Environment; Develop Simulation Environment; Integrate and Test Simulation Environment; Execute Simulation; and Analyse Data and Evaluate Results. Since this work's contributions are focused on improving conceptual analysis, our methodology is based on the first four steps of DSEEP.

2.2. Model Driven Architecture

Model Driven Architecture (MDA) is a software development standard proposed by Object Management Group (OMG) that presents a software development guideline using different abstract representations (models) of the final software. MDA defines three types of abstract models: i) The Computation-independent Model (CIM) conceptually defines the system without specifying computational aspects, such as the separations of functionalities in different sub-systems; ii) The Platform-independent model (PIM) also conceptually defines the system, taking into account the computational aspects involved, providing a view that shows the general structure of the system. However, the PIM model does not show the specific technologies used in different parts of the system; iii) The Platform-specific model (PSM) defines the system, considering the computational aspects and specifying the different technologies used within said system. From its abstract models, MDA promotes model-driven development, allowing the automatic creation of systems from well-defined models.

2.3. Object-Process Methodology

Object-Process Methodology (OPM) [Dori 2002] is a systems modeling language specified as ISO/PAS 19450. An OPM model is composed of a set of Object Process

[3]QVT language specification available at https://www.omg.org/spec/QVT/

diagrams (OPDs), which comprises objects, processes, states, and links between them. OPDs contain a system's structure and behavior and can have different levels of detail on particular objects and processes. A single object in an OPD can comprise several objects/processes in another OPD. This feature allows the OPM developer to create OPDs with differing levels of abstraction.

The OPM also has a mapping of model elements into English sentences with complete equivalence to its diagrammatic form, called Object-Process Language (OPL), allowing a more direct way to interpret what was modeled. Figure 1 illustrates OPM in practice; it contains an OPD representing a system graphically alongside its equivalent OPL textual representation. An OPM modeler can choose to write sentences using OPL or make graphical models using OPD.

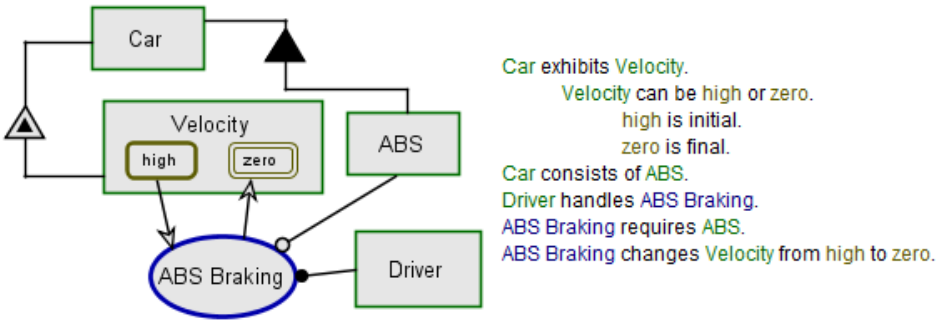

Figure 1. OPM Model Example.

3. OPM-UML based Model-Driven Methodology for DSEEP

This section presents this work's proposed OPM-UML based model-driven methodology. A general view of the methodology is presented in Figure 2 and will be explained throughout this section.

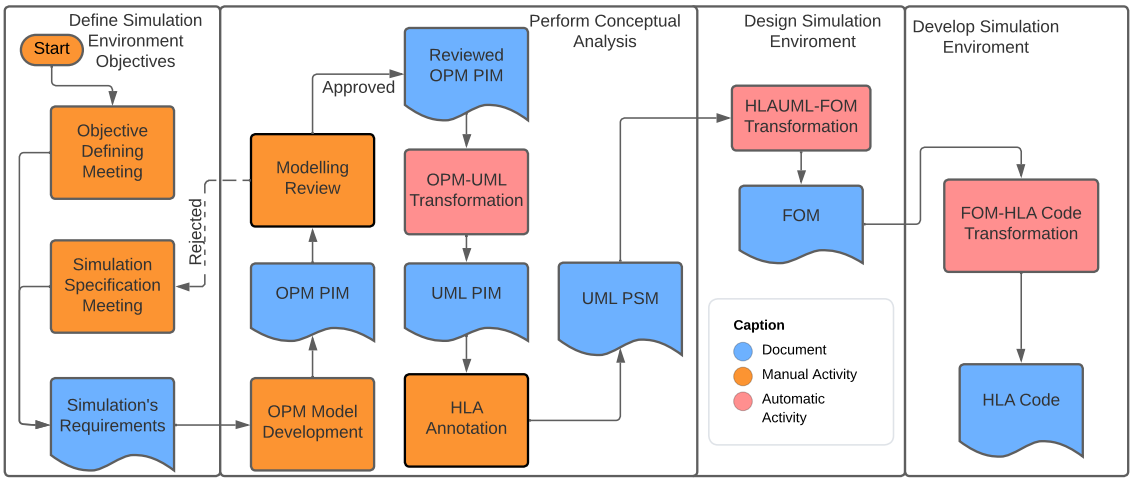

Figure 2. Proposed Development Methodology.

The OPM modeling process is employed at the beginning of the DSEEP "Perform Conceptual Analysis" step. It receives the simulation's requirements from two initial

meetings in the "Define Simulation Environment Objectives" step. One focused on establishing general objectives, and another on specifying how they will be achieved. These meetings are commonly attended by the project's stakeholders, often non-technical people. As such, their requirements are described with a high level of abstraction. Modeling languages like UML and SysML require more than high-level simulation specifications to correctly model the system's structure and behavior, separated into different model types. The use of OPM, which allows for more abstract models, in our methodology aims to bridge the gap in abstraction between the stakeholder's high-level requirements and the need for more information on the lower-level simulation specifications. This solution leads to a better understanding of the stakeholders' needs, making the simulation conceptual modeling phase more flexible and friendly to stakeholders and developers, avoiding errors, and reducing the project's cost.

OPL is perhaps the biggest factor in allowing its diagrams to be readable by non-technical people. The OPL mapping allows for an ease of understanding of the many different types of links connecting processes and objects in a diagram. Thus, it increases the readers' comprehension of a conceptual model. Also, the simulation system's lower-level details can be represented without damaging the high-level view of a model by using different levels of Object Process Diagrams (OPDs). Some OPDs can abstract many aspects of the system in favor of readability, while others OPDs can contain many details of specific parts of the system. Models with many diagram types, like UML and SysML, can have many problems with development and user understanding [Ong e Jabbari 2019]. In summary, the adoption of OPM allows us low and high-level modeling using intuitive graphical diagrams at the same time, and it also allows us to automatically generate OPM elements based on textual input in OPL.

As seen in Figure 2, a special feature of the proposed methodology is the OPM modeling review activity, where stakeholders evaluate the model before development begins. In DSEEP, reviews are normally done after project development, in the "Analyse Data and Evaluate Results" step. Our approach allows the development team to ask questions regarding missing or ambiguous requirements while an abstract view of the system is already being presented. This makes it so the stakeholders can understand more simulation details and give their input regarding doubts about specific requirements. If the developed model requires many changes to its high-level views, it is deemed "rejected," and the meeting of the specific objectives is again held. If the developed OPM model requires only a few changes to its high-level views, these changes should be made (in addition to completing the lower-level OPDs with the information gained during the review), and the model is deemed "approved" to continue to the next activity.

Once the OPM model has been approved, the next step is transforming the PIM into a PSM. UML class diagrams were chosen to model the PSM because treating all of the OPM objects as classes with their own types makes it easier to change the class type to fulfill a specific HLA role while maintaining the rest of the class intact. Only class diagrams are used because the code generation receives FOM files, which are strictly structural, so there is no need for the intermediate models to have diagram types containing behavioral information. Transforming the OPM PIM into a UML PSM is a process that requires two activities. First, the automatic OPM-UML model-to-model transformation creates a UML PIM. Then, a manual model annotation step is performed

to specify the HLA roles (“Federate,” “Object,” or “Interaction”) of certain classes from the UML PIM, creating the UML PSM (nicknamed HLAUML).

With the HLAUML model, the automatic HLAUML-FOM transformation can occur. This transformation generates the FOM containing all data types, object/interaction classes, attributes, and hereditary relations. The final step is the HLA code generation. This transformation involves reading the input file and the language generation templates and generating a usable HLA API from which the developer can build the federate.

Lastly, this methodology also maintains the possibility of solely using UML if that is more convenient for the team developing the simulation. The UML model would be developed, reviewed by the stakeholders, and then annotated. Afterward, it would follow the same path as Figure 2.

4. Implementation for the proposed Methodology

The proposed methodology was implemented as depicted in Figure 3. Initially, the OPCat [Dori et al. 2010] tool is used for the OPM modeling activity and for the OPM-UML transformation. Then, a Java program was developed to facilitate model annotation by providing an interface where the developer can choose which HLA role to give to each class of the input model. This program also automatically runs programmed QVT-Operacional model-to-model transformations to turn the annotated UML model into the federation FOM. Lastly, the resulting FOM is then used by our developed Code Generator to generate the final HLA code for the federation as it was specified by the FOM.

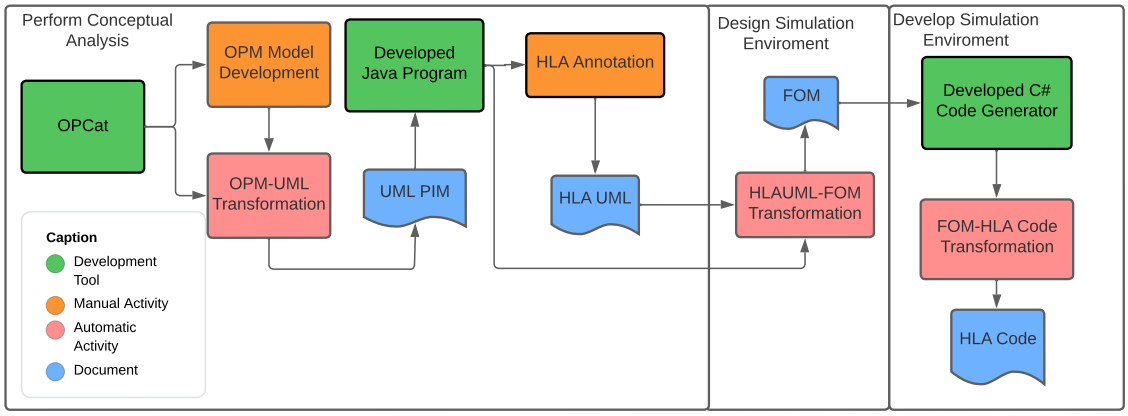

Figure 3. Implementation of the Proposed Methodology.

5. Case Study

This section showcases a case study performed in order to demonstrate how OPM is easily understood by people unfamiliar with it or other conceptual languages. In this case study, a military software maintainer with only basic notions of conceptual languages and no prior knowledge of OPM was tasked with developing an OPM model to represent a federation connecting two real simulators used by the Brazilian Army to educate two distinct sectors. This model would then be analyzed to determine if it accurately depicts the simulation and to see what someone unfamiliar with conceptual languages can do with OPM. We highlight that UML model and other developing tasks are automatic derived

from OPM model and are omitted on this section. The case study used two real simulators here called *STat* and *SCon* and they have the following characteristics.

STat Simulator: i) A virtual tactical simulator developed to educate the people in charge of giving orders to the operators of a specific artillery battery (*STBattery*); ii) It represents a virtual environment where *STBattery* can be controlled and have its specific operations be performed to execute missions; iii) It offers a visual and interactive representation of the artillery systems, containing a 2D view of the terrain where military units are represented by 2D symbols and a 3D view of the terrain with a movable camera that represents the vehicles of *STBattery* with detailed 3D models; iv) Trainees use this simulator to learn the doctrine required to use *STBattery* to perform missions and evaluate the damage done to the targets.

SCon Simulator: i) Represents the overall environment of the military operation, including allied and enemy forces, marked areas, movement routes, and other relevant factors; ii) Contains information regarding the terrain's topography and updates the positions and statuses of the deployed units to reflect their actions during the operation. iii) Does not have a personalized doctrine for the operation of *STBattery* (since it contains various other types of units). Those details are abstracted in favor of only representing the final effects on the targets that *STBattery* shot.

The first activity of this case study was the "Objective Defining Meeting". This meeting was held, and the main objective of this federation was established:

> The main objective of the distributed simulation is to integrate the simulators *SCon* and *STat* and enable military exercises involving *STBattery* to be executed in parts by *SCon* and *STat*.

Furthermore, the following directives were solidified:

- Doctrine-specific actions regarding *STBattery* will be executed by *STat* to provide the artillery commander trainees with a more immersive and realistic experience.
- Relevant data regarding the state of the mission, like target coordinates and current battery position and where the launched munitions landed, shall be shared between the simulators.
- Both simulators shall use the Real-Time Platform Reference (RPR) FOM [Möller et al. 2014] as a basis for the distributed simulation communication. Extending the FOM when it is deemed necessary to fulfill a requirement.

Following the "Objective Defining Meeting," the "Simulation Specification Meeting" was held, and a list of requirements was created. These requirements relate to functionalities the simulators need to implement to be used during the simulation. The Table 1) summarizes the requirements. Most classes present in the requirements are from the RPR FOM specification.

From these requirements, the subject of this study (stakeholder that is not a developer) was tasked with creating an OPM diagram containing all of the requirements related to the direct communication between the simulators. The resulting OPDs can be seen in figures 4, 5, and 6.

Initially, the stakeholder had some difficulties coming up with the main diagram, as it was their first experience with OPM. However, after looking at some basic examples

Table 1. Summarized requirements of the distributed simulation between *SCon* and *STat*.

RQ	Description
RQ1	*SCon* should be Time Regulating. This means publishing the *Time* and *TimeScale* interactions.
RQ2	*SCon* shall publish the allied and enemy units involved in the exercise with the *AggregateEntity* RPR ObjectClass.
RQ4	*SCon* shall publish the *Drawings* and *DrawingLayers* that are present in its exercise.
RQ5	*SCon* shall publish the impact of munitions from units owned by it with the *MunitionDetonationClass* RPR InteractionClass.
RQ6	*SCon* shall subscribe to *MunitionDetonation* interactions sent by *STat*. Damaging any of its entities that are within the radius of the detonation.
RQ7	*SCon* shall be able to acquire and release the ownership of units (*AggregateEntity* RPR object) from/to *STat* using the RPR *TransferControl* interaction class.
RQ8	*STat* shall be Time Constrained, meaning it shall subscribe to the *Time* and *TimeScale* interactions and change its date/time and simulation speed accordingly.
RQ9	*STat* shall subscribe to the *AggregateEntity* objects published by *SCon*, representing them in the 2D terrain view with symbols corresponding to its *EntityType* and in the 3D view as either blue or red cubes depending on whether they are an ally or an enemy.
RQ10	*STat* shall publish updates to the attributes of *AggregateEntity* objects it has ownership of.
RQ11	*STat* shall subscribe to the *Drawing* and *DrawingLayer* interactions, displaying the drawings in the 2D view of the terrain and organizing them into their respective layers.
RQ12	*STat* shall be able to acquire and release the ownership of *AggregateEntity* objects of type *SBattery* from/to *SCon*; Enabling *STat* to control the *SBattery* and perform the doctrine actions to execute the artillery mission and release the control of it when the mission is complete.
RQ13	*STat* shall publish *MunitionDetonation* interactions to indicate all detonation points of munitions launched by *SBattery*.
RQ14	*STat* shall subscribe to *MunitionDetonation* interactions and be capable of calculating and applying damage to vehicles of a given *SBattery* (that it has ownership over) in the radius of the detonations.

of OPM diagrams for various purposes, the stakeholder was able to come up with a suitable design for the OPD of Figure 4 and thus modeling the matter in which the simulators would send and receive data from each other. Figures 5 and 6 contain the diagrams responsible for representing all of the requirements from Table 1. Thus, it contains the objects and interactions that each simulator publishes/subscribes to and a simplified behavior of how the data from a published class goes to the simulator that subscribed to it. When modeling these diagrams, the stakeholder noted that the OPL representation of the model (automatically generated from OPCat) greatly assisted them in confirming the semantics of the connections between the diagram elements.

One interesting point about this model is that it does not have explicit subscribing processes for the simulators. Rather, they receive information directly from the publishing process of the other simulator. When asked about this design choice, the stakeholder reported that for every requirement requiring one simulator to publish a class, there was another for the other simulator to subscribe to. So, they found it appropriate to leave the subscribing implicit. This is a valid way to model a federation where everything published by one simulator is subscribed by the other.

This case study showed that an individual with next to no prior knowledge of OPM

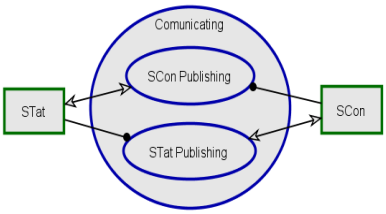

Figure 4. Main OPD of the Subject Model

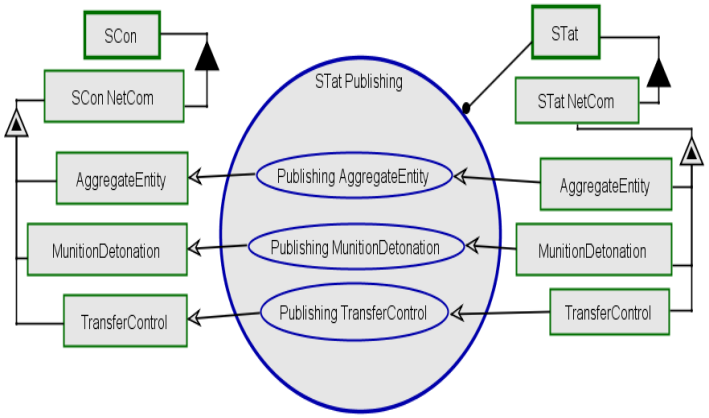

Figure 5. Subject Model: STat Publish OPD

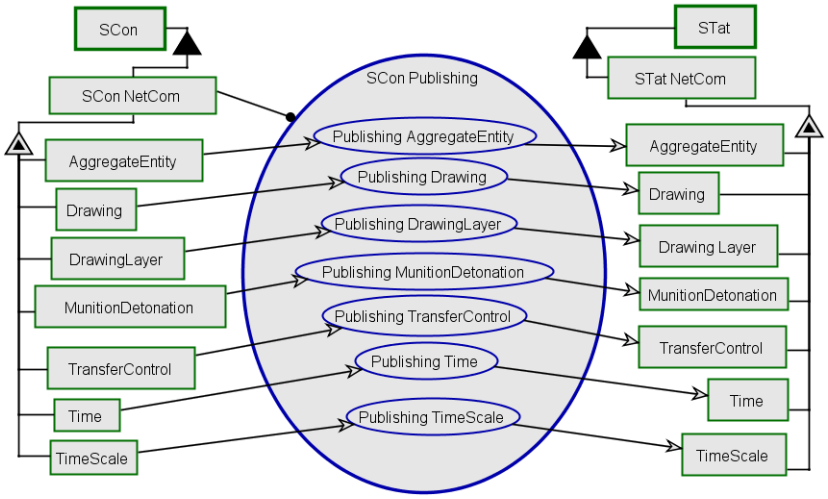

Figure 6. Subject Model: SCon Publish OPD

could still model distributed simulation requirements, meaning OPM is quite learnable and understandable for people with no prior contact with it. In addition, as explained by the case study, from our methodology, the stakeholder could be the one to work on the OPM conceptual modeling, and after, the developers can automatically generate a UML model, FOM, and HLA source code, as shown in Figure 3.

6. Related Works

This section presents related works that have developed similar approaches to this paper's proposition.

The work described in [Dere et al. 2020] proposes a framework for agent-based simulations that explores SysML as the conceptual modeling language. Specifically, they explored Block Definition Diagrams (BDD) and Internal Block Diagrams (IBD) alongside the Acceleo tool for code generation. SysML was chosen due to its popularity and its purpose of being specialized in systems modeling. Unlike ours, they did not focus on HLA code generation from a MDA approach.

A Model Driven Engineering (MDE) method based on formal models to develop distributed simulations for healthcare systems is proposed in [Arronategui et al. 2020]. The authors use a Discrete Event System (DES) to determine disease propagation, and the

method starts with creating UML Activity diagrams, which are then transformed into Petri net hierarchical models. Afterward, the Petri net models go through structural analysis and are finally transformed into source code. Different from our work, which offers a more readable model (OPM) in the first step of MDE, they look at the problem from the perspective that the simulation modeling requires the use of mathematical models that describe the emergent behavior without the need to describe low-level details.

A distributed simulation development approach that unites DSEEP and MDA, called Model-Driven DSEEP (MoDSEEP), is presented in [Bocciarelli et al. 2019b]. MoDSEEP adds automatic transformations for DSEEP steps until source code generation. The first conceptual model is made in SysML following the PIM model format, and then it is annotated with HLA-specific terms, like "Federation," and "ObjectClass," etc, becoming a PSM. The PSM is then transformed into the simulation's FOM and source code. Our work differs from [Bocciarelli et al. 2019b] in key parts, such as the choice of modeling language(s) and the number of model transformations. We use two modeling languages: OPM for the initial high-level PIM and UML for representing the system with more details, including adding HLA concepts. We also adapted the number of model transformations by adding one more model to model transformation, from OPM to UML. Another important difference is how we handled UML/SysML profiles. Instead of annotating HLA concepts into the models, we use Ecore meta-models to validate and annotate our models.

In summary, our work has the benefit of generating HLA simulations for general use while also having multiple modeling languages (OPM and UML) to help with mutual understanding with stakeholders.

7. Acknowledgement

We thank the Brazilian Army and its Army Strategic Program ASTROS for the financial support through the SIS-ASTROS GMF project (TED 20-EME-003-00).

8. Conclusions

This work presented a new conceptual modeling step integrated into a model-based DSEEP process. Our approach results in a new model-driven methodology for HLA distributed simulation developments that use OPM in the initial modeling phase to increase the alignment between non-technical stakeholders and the development team. The approach allows game developers to have a clear way to implement HLA compatibility into their games. The methodology uses UML as an intermediary language created automatically from the OPM model. UML is used for deep specification to generate the federation's FOM. An implementation was also provided, demonstrating its feasibility. Experiments demonstrated how OPM contributes to representing the simulation's behavior through readable diagrams while offering a way to model-to-model transform to UML. The advantage of the newly proposed conceptual modeling step is that the provided methodology and its implementation can generate usable HLA source code for a federation initially modeled in a learnable and understandable high-level model.

References

Arronategui, U., Bañares, J. Á., e Colom, J. M. (2020). A mde approach for modelling and distributed simulation of health systems. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*, pages 89–103. Springer.

Basnet, S., Valdez Banda, O., Chaal, M., Hirdaris, S., e Kujala, P. (2020). *Comparison of system modelling techniques for autonomous ship systems*, pages 125–139.

Bocciarelli, P., D'Ambrogio, A., Falcone, A., Garro, A., e Giglio, A. (2019a). A model-driven approach to enable the simulation of complex systems on distributed architectures. *SIMULATION*, 95(12):1185–1211.

Bocciarelli, P., D'Ambrogio, A., Giglio, A., e Paglia, E. (2019b). Model-driven distributed simulation engineering. In *2019 Winter Simulation Conference (WSC)*, pages 75–89.

Choi, C., Seok, M.-G., Choi, S. H., Kim, T. G., e Kim, S. (2013). Serious game development methodology via interoperation between a constructive simulator and a game application using hla/rti. In *International Defense and Homeland Security Simulation Workshop*.

Crues, E. Z., Dexter, D., Falcone, A., Garro, A., e Möller, B. (2022). Spacefom-a robust standard for enabling a-priori interoperability of hla-based space systems simulations. *Journal of Simulation*, 16(6):624–644.

Dere, B. E., Görür, B. K., e Oğuztüzün, H. (2020). A framework for constructing agent-based aerospace simulations using model to text transformation. In *Proceedings of the 2020 Summer Simulation Conference*, pages 1–12.

Dori, D. (2002). *Object-Process Methodology*. Springer Berlin, Heidelberg.

Dori, D., Linchevski, C., Manor, R., e Opm, O. (2010). Opcat–an object-process case tool for opm-based conceptual modelling. In *1st International Conference on Modelling and Management of Engineering Processes*, pages 1–30. University of Cambridge Cambridge, UK.

dos Santos, G. e Nunes, R. (2022). An approach to build source code for hla-based distributed simulations. In *Anais Estendidos do XII Simpósio Brasileiro de Engenharia de Sistemas Computacionais*, pages 98–103, Porto Alegre, RS, Brasil. SBC.

D'Ambrogio, A., Falcone, A., Garro, A., e Giglio, A. (2019). Enabling reactive streams in hla-based simulations through a model-driven solution. In *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–8.

Graham, J. (2007). Creating an hla 1516 data encoding library using c++ template metaprogramming techniques. In *2007 Spring Simulation Interoperability Workshop Proceedings, 07S-SIW-035*.

IEEE (2010). Ieee standard for modeling and simulation (m&s) high level architecture (hla)– federate interface specification. *IEEE Std 1516.1-2010 (Revision of IEEE Std 1516.1-2000)*, pages 1–378.

IEEE (2022). Ieee recommended practice for distributed simulation engineering and execution process (dseep. *IEEE Std 1730-2022 (Revision of IEEE Std 1730-2010)*, pages 1–74.

Lee, T.-D., Yoo, S.-H., e Jeong, C.-S. (2005). Hla-based object-oriented modeling/simulation for military system. In *Systems Modeling and Simulation: Theory and Applications: Third Asian Simulation Conference, AsianSim 2004, Jeju Island, Korea, October 4-6, 2004, Revised Selected Papers 3*, pages 122–130. Springer.

Lees, M., Logan, B., e Theodoropoulos, G. (2006). Agents, games and hla. *Simulation Modelling Practice and Theory*, 14(6):752–767. Distributed Systems Simulation.

Mall, R. (2018). *Fundamentals of software engineering*. PHI Learning Pvt. Ltd.

Möller, B., Dubois, A., Leydour, P., e Verhage, R. (2014). Rpr fom 2.0: A federation object model for defense simulations.

Möller, B., Karlsson, M., e Löfstrand, B. (2006). Reducing integration time and risk with the hla evolved encoding helpers.

Ong, D. e Jabbari, A. (2019). A review of problems and challenges of using multiple conceptual models.

Petty, M. D. e Windyga, P. S. (1999). A high level architecture-based medical simulation system. *Simulation*, 73(5):281–287.

Possik, J., Zacharewicz, G., Zouggar, A., e Vallespir, B. (2023). Hla-based time management and synchronization framework for lean manufacturing tools evaluation. *SIMULATION*, 99(4):347–362.

Šenkỳr, D. e Kroha, P. (2021). Problem of inconsistency in textual requirements specification. In *Proceedings of the 16th international conference on evaluation of novel approaches to software engineering–ENASE*, pages 213–220.