Distributed simulation for a modeling and simulation tool: PAPYRUS

Simon Gorecki, Judicaël Ribault, Grégory Zacharewicz, Nicolas Perry, Yves Ducq

DISTRIBUTED SIMULATION FOR A MODELING AND SIMULATION TOOL: PAPYRUS

Simon Gorecki
IMS lab.
University of Bordeaux
351 Cours de la Libération
33400 Talence, France
simon.gorecki@u-bordeaux.fr

Judicael Ribault
IMS lab.
University of Bordeaux
351 Cours de la Libération
33400 Talence, France
jribault@gmail.com

Gregory Zacharewicz
LG2IP Lab.
IMT – Mines Ales
6 Avenue de Clavières
30319 Alès, Cedex, France
gregory.zacharewicz@mines-ales.fr

Nicolas Perry
Arts et Métiers ParisTech
ENSAM of Bordeaux
Esplanade des Arts et Métiers
33400 Talence, France
perry.nicolas@ensam.eu

Yves Ducq
IML lab.
University of Bordeaux
351 Cours de la Libération
33400 Talence, France
yves.ducq@u-bordeaux.fr

ABSTRACT

Modeling and Simulation (M&S) are important steps in design process. Increasing the complexity of engineered system tend to raise Distributed Simulation. Papyrus, an open source UML/SysML modeler of the Eclipse foundation provides a tool to model and simulate these two languages thanks to the fUML standard. However, Papyrus is not yet able to deal with Distributed Simulation. In this paper, we propose a Distributed Simulation composed of several Papyrus instances. Each made of an UML Profile, a Moka extension, and software architecture. The main objective is to synchronize the execution of these Papyrus instances. We are using Functional Mockup Interface (FMI) to enable communication between them. Nevertheless, the FMI standard does not natively include time management rules. Indeed, we propose to use the High Level Architecture (HLA) standard to handle time management between Papyrus simulations instance, through FMI communication.

Keywords: Model Driven Architecture; HLA; FMI; Papyrus; Simulation.

SpringSim-Mod4Sim 2019, April 29-May 2, Tucson, Arizona, USA;

1 INTRODUCTION

In industry, the design of complex systems requires Modeling & Simulation (M&S) steps to represent behaviors and interactions in the system and between systems. As technologies are growing, the systems complexity also increases and makes systems more difficult to model and simulate. Along with this growing complexity, risks, hazards, and threat must also be considered during the modeling process. Several researches which prove this fact have been done about project risk management (Altuhhova, Matulevicius, and Ahmed 2013; Better et al. 2008). In this context, it is required to divide the complexity of a system using Distributed Simulations (DS). However, industrial context providing the application case is using UML and Papyrus that are not ready for DS.

Papyrus, an open source UML / SysML modeler, provides to users and developers a tool to specify UML models and then to execute them thanks to the Foundation UML Subset (fUML) standard (OMG Publication 2018). Papyrus is already used by our partner company to model and simulate the production of solar power plant. This software is used thanks to its user friendly modeling and simulation features according to the fUML standard. We can also describe specific behaviors according to the UML profile. Those profiles can be taken into account during the simulation run thanks to the Moka execution engine. However, as introduced before, growing complexity implies many risk and hazard impacts on models. Modeling all these risks would be very complex and make the initial model unreadable (Gorecki et al. 2018).

In order to solve this issue, (Gorecki et al. 2018) first built an extension of the Moka execution engine to handle risk and hazard outside the model with a Functional Mock-up Unit (FMU). The FMI standard allowed us to relocate and factor complex tasks without overloading models. In the following of these works, to deal with risks management in M&S, we proposed to divide complexity into several DS components. The purpose of this paper is managing Papyrus instances as DS components to allow users and developers to deal with complexity and time constraints. In details, after having managed interaction with the Moka scheduler in previous works, the goal is now to implement synchronization points mechanism in Moka. This is a complex problem since it implies defining a synchronization, communication algorithm which is not defined by the FMI Co-Simulation standard.

To solve the time related issues in distributed context, we use the High Level Architecture (HLA) standard (IEEE Computer Society 2010a, 2010b, 2010b), one of the most used standard in DS. Indeed, in HLA, the Run Time Infrastructure (RTI) can solve interoperability issues between components, provide communication mechanism and time management. Hence, to create a Papyrus distributed simulation, we propose to link HLA and the Moka engine through FMI. We have already coupled Moka and FMI in our previous paper, and some research already exist on coupling FMI and HLA (Bouanan et al. 2018) and (Yilmaz et al. 2014). In short, the main purpose of this contribution consist in merging these two standards, HLA RTI as scheduler for several Papyrus instances and FMI for compatibility with other components.

2 BACKGROUND

This section introduces the different concepts used in this paper for the proposed solution. The first subsection is dedicated to the Papyrus tool and its execution engine: Moka. Then, the Functional Mock-up Interface standard (FMI) is described. Finally the High Level Architecture (HLA) standard is explained in 2.3 section.

2.1 Papyrus & Moka execution engine

Papyrus is the UML/SysML modeler of the Eclipse foundation. UML allows the user several modeling notations to represent structures and behaviors of its system. This standard is however not very efficient for system description. That is the reason why the UML profile exists. Created by the Object Management Group (OMG), a UML profile allows users to specify its own UML language on its own semantic objects.

Papyrus provides tools for modeling, and executing these two standards. It is based on Eclipse and is an open source project. The execution part is handled by MOKA, a fUML execution engine. In accordance with its primary goal to implement the complete standard specification of the UML2, Papyrus provides an extensive support for UML profiles.

Moka is a Papyrus plugin designed to provide an execution environment complying with the OMG standards UML. The Foundational UML Subset (fUML) is an executable subset of the UML standard which can be used to define, in an operational style, the structural and behavioral semantics of systems in order to be simulated. (Guermazi et al. 2015). With this standard, Moka is able to execute UML models designed under the Papyrus UML editor. Moreover, designed for Papyrus, Moka is open source and can be extended to support the UML Profile or alternative execution semantics, and thereby, be adapted to multiple usages.

2.2 Functional Mockup Interface (FMI)

The Functional Mock-up Interface (FMI) for Co-Simulation interface is designed to simulate tools coupling and subsystem models coupling. In both cases, coupled systems are exported by their own tools. FMI is a standard interface for time dependent coupled systems which are continuous in time or time-discrete (Bastian et al. 2011; Blochwitz 2016; Sievert 2016). It provides interfaces between master and slaves and addresses both data exchange and algorithmic issues. FMI for Co-Simulation consists of two parts:

- Co-Simulation Interface: a set of C functions for controlling the slaves and for data exchange of input and output values as well as status information.
- Co-Simulation Description Schema: defines the structure and content of an XML file. This slave specific XML file contains “static” information about the model (input and output variables, parameters …) and the solver/simulator (capabilities …). The capability flags in the XML file characterize the ability of the slave to support advanced master algorithms which use variable communication step sizes, higher order signal extrapolation etc.

A component implementing the FMI is called Functional Mock-up Unit (FMU). It consists in one zip file containing the XML description file and the implementation in source or binary form (dynamic library). A master can import an FMU by first reading the model description XML file contained in the zip file. Coupling simulators by FMI for Co-Simulation hides their implementation details and thus can protect intellectual property.

2.3 High Level Architecture

HLA defines a framework which allows the creation of a Distributed Simulation (DS). DS participants are called federates, they can communicate with each other. The standard was originally created by the Office of Defense Modeling and Simulation (DMSO) of US Department of Defense (DoD) to facilitate the assembly of stand-alone simulations with a different architecture. The original goal was the reuse of military applications, simulations, and sensors. This standard is designed to resolve interoperability issues and reusability between software components. Another interesting aspect of this specification is the synchronization aspect. It allows the dynamic management of interoperability issues with simulations exchange messages: it must be ensured that messages are sent at the right time, in the right order, and that they do not violate causal constraints. To do so, various systems for synchronization of processes and time management are proposed by HLA.

According to the HLA standard, all simulations participating to the application are called "federate". A classical HLA federate consists of a simulation model and local RTI component (LRC). The simulation model is a physical, mathematical, or logical representation of processes and systems. These entities can communicate with each other through a Run-Time Infrastructure (RTI). It is the RTI which manages the federation, authorizes federates to communicate or not, and provides various services such as time management, file or data exchange, etc. The FOM file is a XML file which describes interactions/communications between federates.

3 STATE OF ART

HLA and FMI are well known standards of Co-Simulation and Distributed Simulation. They both are described in different specification books. For HLA, (IEEE Computer Society 2010a) describes how interactions can be done between each component of de DS: Exchange mechanism are done by Subscribe (read) and Publish (write) provided by the RTI. Those communications are described by SOM and FOM XML files. However, there is no simple way to describe the federate behavior inside the federation: each federate is autonomous.

HLA imposes an architectural rule for its federates. Each one of them has to assume its own time management. HLA architecture can handle three advancing time mechanisms for each federate member of the federation.

- Coordinated time, each federate advancing time is coordinated with each other
- On messages, when the advance in time of a federate is triggered by reception of a message over HLA communication mechanism
- Optimistic, where each federate can advance in time at its own speed independently from the others. Often used when the federate´s time is real time. If not, it receives a message from the past and triggers a rollback of the simulation.

In literature, authors worked on merging the HLA and the FMI standards. Yilmaz et al. presented in their paper (Yilmaz et al. 2014) a method to automatically generate an FMU entity able to join an HLA federation as member. A wrapper reads the FMU specification and generates the HLA layer. To demonstrate the FMUFd (FMU-Federate) usage, authors developed a simple DS example with MAK HLA RTI. In this paper, time synchronization between the two standards (FMI & HLA) is solved by updating the federate time at each running step of the FMU model.

Authors presented in (Bouanan et al. 2018) an application running several federates connected to the same RTI and communicating together. Among these federates, one of them is a hybrid federate hosting an FMU making a communication bridge between HLA classical federates and a hybrid HLA FMI element.

In literature, efforts has been done to upgrade the FMI standard by adding extensions and enable treatment of discrete-time models (Franke et al. 2017). In this paper, authors proposed a solution to synchronize in time FMUs with others elements by adding a clock in the model description and implement interface to call it. An FMI export with synchronous features was implemented into the tools Dymola and OpenModelica. An FMU import was implemented in the optimization solver HQP.

Other papers are also interesting concerning use of Papyrus software. (Guermazi et al. 2015) discuss about the fUML and the Alf standards available in Papyrus which allow to execute UML models. Authors of this paper highlight four major concerns regarding the design of fUML execution engine: Extensibility, control and observability, time support, and connectivity with external tools.

In (Ellner et al. 2011), authors highlight the limits of the fUML standard by explaining is lack of support for distributed execution in order to guide and support team members with their activities. FUML also requires explicit modeling of auxiliary user specific attributes and behavior of model elements, which is a repetitive and error-prone task. The author presented in this paper fUML extensions able to support distributed execution and human interactions. With these FUML extensions it becomes feasible to enact reusable standard software process models in realistic projects.

In (Gorecki et al. 2018), authors proposed an entry point in the Moka execution engine allowing control on local scheduler. This paper demonstrates how to interact with time management during an execution process in order to simulate hazards. This Moka extension allows to outsource risk management in an independent tool in order to connect an FMU during the simulation execution process and outsource tasks. This concept will be more explained in the next section.

4 CONTRIBUTION

The contribution in this paper is based on two previous works recalled in state of the art. The first one is about controlling simulation time in Papyrus with a specific Moka extension interacting with an FMU file (Gorecki et al. 2018). The second one is about creating an HLA federate hosting a Hybrid HLA/FMI federate able to communicate with each other (Bouanan et al. 2018). Merging these two contributions consists in defining a new Moka extension able to communicate with another instance of Papyrus, allowing communication and time synchronization through FMI and HLA mechanism. Section 4.1 is recalling previous works to situate the contribution of this paper, while the contribution is detailed in section 4.2 and 4.3.

4.1 Related contributions

The main objective of this paper is to make a Papyrus simulation able to communicate with others Papyrus instances at runtime. This contribution is based on a previous paper (Gorecki et al. 2018) where we proposed a Moka extension able to communicate with an external tool (see Figure 1). This tool had to generate hazards and insert them into the Papyrus simulation process. This previous paper demonstrated that we can use an external tool to influence Papyrus simulation time.

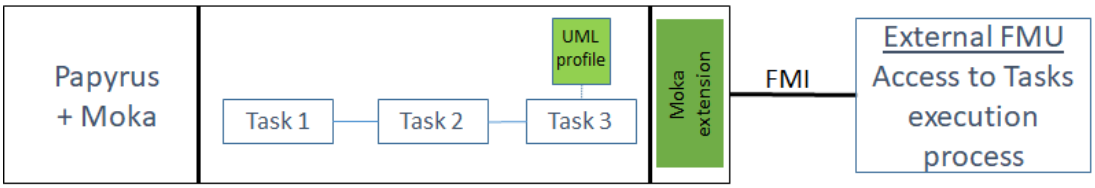

Figure 1 : Papyrus + Moka extension connected to FMU

For outsourcing risks management, we had to develop (1) a UML profile and (2) a Moka extension.

An UML profile is a new semantic layer added over the UML language standard. It allows developers to add semantic values over the original model.

The Moka extension has access to the content of our current custom UML profile, as well as the whole simulation. In this extension, we are able to define (with java code) specific behaviors according to what is given by the UML profile. In Figure 1 above, the Moka extension makes a request to an external FMU. This FMU can generate hazards and delay tasks depending on parameters which are the content of the UML profile. Figure 2 describes a simple example of how the UML profile and the Moka extension can affect a simple model. The left sequence flow diagram shows the normal execution of Figure 1 without applying our Moka extension: each task is executed according to its "normal" time (normal time is the time affected to a task in fUML Papyrus model). In the right side of the figure, we applied our UML profile on Task 3 (as in Figure 1). We can see when Task 3 is executed, the Moka extension sends all the information contained in the UML profile to the FMU. The FMU determines (based on the information contained in the UML profile) if this task should be delayed or not, and if yes, for how long. We can see on the right sequence flow diagram that the Moka extension extends the simulation time of Task 3 to 3 units of time.

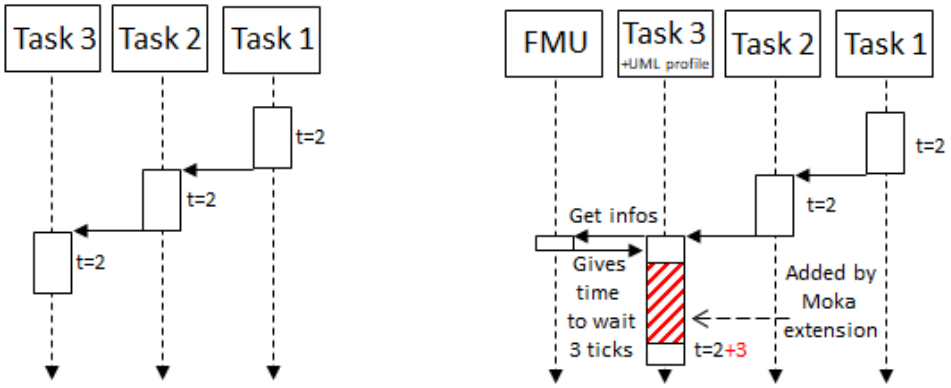

Figure 2 : Sequence flow diagram of Papyrus execution VS Papyrus execution with custom risks management profile

In this example, the FMU determine delay time according to content of the UML profile (name of the task, simulation time, etc) and use databases, web services, and statistics algorithme to determine the delay.

(Bouanan et al. 2018) presents a hybrid HLA/FMU connected to an RTI, which communicates with an external application through the FMI standard. In our paper, we use the same approach: a hybrid HLA/FMU component able to communicate with one local Papyrus execution and a HLA federation.

4.2 Hybrid HLA/FMI federates to manage Papyrus instances

The objective here is to run several Papyrus instances, each one having its own hybrid HLA/FMU component. We use the FMI standard as entry point in the Papyrus simulation, and the HLA standard for its functionality of time management, and communication protocols. The main objective is to implement waiting points between Papyrus instances. Using a hybrid component brings us several advantages. First, all the development which we are doing are be compressed in one single FMU file. Indeed, it is easy to reuse our proposition in futur Papyrus project directly on importing the file. Also, by assuming the FMI standard, we can connect other compliant tools in industry. And finally, using the HLA standard enable connection of other simulations to the final DS.

When an instance of Papyrus has to wait for another instance of Papyrus, messages are exchanged amongst them through HLA communication mechanism. At the beginning of the federation, each federate subscribes to an interaction class “Message” (described in Figure 2) in order to be informed of every state modification during the global distributed simulation. HLA mechanism does not allow a federate to address message directly to a recipient. Federates can only publish and subscribe Objects or Interactions. At the initialization of the federation, each federate subscribes to the same interaction class (Figure 4). Indeed, during the simulation, each federate receives all interactions. They use “src” and “dst” attributes to identify if there are concerned by the message or not. If not, they simply ignore the message. If yes, they can publish an answer. When an instance of Papyrus enter in a synchronized state, the associated federate publishes an interaction in order to inform the other members.

Each Papyrus instance has the same UML Profile and the Moka extension corresponding to the Figure 3 below.

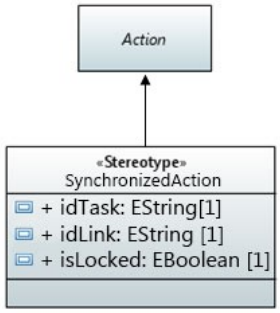

Figure 3 : Custom UML Profile

Every synchronized Task (Action in UML syntax) inherits from "*SynchronizedAction*" Stereotype which contains:

- *idTask*: a unique identifier. It will allow any federate to identify where it comes from. For example, P1.3 is the id of the Task 3 in the federate 1.
- *idLink*: id of the associated locker/unlocker task.
- *isLocked*: a Boolean variable that determine if it is a locking task or an unlocking task.

At the execution of a synchronized task, the Moka extension shares with the hybrid HLA/FMI federate information contained in the profile. The federate behaviors depend on the status of the system.

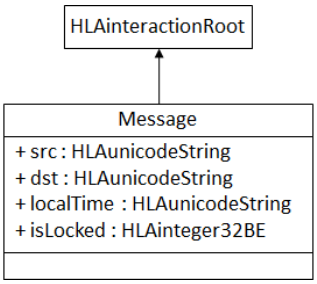

Figure 4 : HLA FOM file

Figure 4 above correspond to the content of the Federate Object Model (FOM) file. It describes every interactions which can occur between federates.

- *src* contains the unique id of the message transmitter. If it is a "lock" or "unlock" message, it contains something such as "P1.3". If it is a reply to a lock message, it only contains its Papyrus id: "P1".
- *dst* contains the target id of the message. With the HLA subscription and publication mechanism, it is impossible to address a message to a specific person. So the federate is able to filter which messages are for them and which are not (if not, ignore them).
- *localTime* is given by the Moka extension. When a lock message is received, the recipient must reply by its own local time.
- isLocked is about the semantic of the message. 0 corresponds to a unlocking message. 1 corresponds to a locking message. 2 corresponds to a locking reply (the federate has to give its Papyrus local time).

During the simulation, each federate has the opportunity to send:

- a “lock message” when Papyrus enters into a synchronized point which needs an unlocking from another Papyrus process.
- an “unlock message” when Papyrus enters into a synchronized point which unlocks another Papyrus process.
- a “time reply message” when a federate receives a notification from a locked Papyrus process, it has to reply its actual local time.

From a technical point of view, we are using in this application the HLA Run Time Infrastructure (RTI) Portico, an Open Source RTI HLA 1516e compliant. The hybrid HLA/FMU works as a time regulating and time constraining federate. On the Papyrus instance side, the time is managed by the DEScheduler of the Moka engine.

The interactions between the two technologies can be divided into two parts. The first part is when Papyrus instance must receive a message. The second part is when it must send a message. At each nanosecond of the simulation, a function is called by the time scheduler to check if nothing commits in the event buffer. We added a function which check modifications in the FMU file outputs. If any modification is noticed, a new event is triggered in order to treat this event. If the Papyrus simulation has to send a message, the Moka extension asks to DEScheduler the local time and get in return the simulation time in nanoseconds after a conversion to milliseconds. This time is given to the FMU with the doStep() function.

4.3 Time management between Papyrus instances

The time synchronization between Papyrus instances is an important point. When an instance of Papyrus is locked by another one, its local clock must continue to fly. This waiting time must be simulated by the Moka extension. The federate interactions goal is to calculate this duration. Figure 6 illustrates a simple example of this time synchronization mechanism.

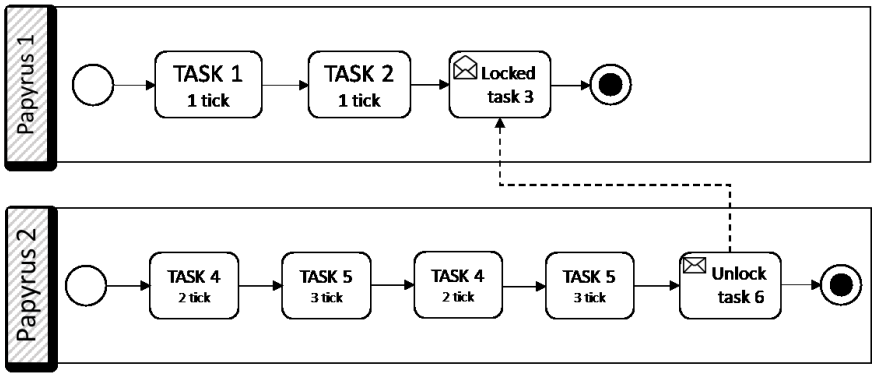

Figure 5 : Simple example of two Papyrus in BPMN

Here are two Papyrus instances which are interconnected by a synchronization point. They both have their own local time, and are completely autonomous until they reach the synchronization point (locked Task 3 and Unlock Task 6 of both process). Moreover, Papyrus instances 1 and 2 do not start at the same time, this delta t is visible on Figure 6. We can see that Papyrus instance 1 (P1) needs to wait for a message from P2 in order to achieved its local process. The synchronization point did not have a duration, in synchronized task P1.3, P1 must wait for an unknown number of ticks, so Moka cannot anticipate how much time this lock costs to P1. Hence, we need to calculate this duration and give it to Moka in order to observe the “lost time” in the P1 simulation.

Figure 6 explains the main exchanges between the components of the simulation. In our example, each Papyrus instance has two normal tasks (e.g. P1.1, P1.2 & P2.4, P2.5), and two synchronized tasks (P1.3 & P2.6). Those two last tasks extends a custom UML profile.

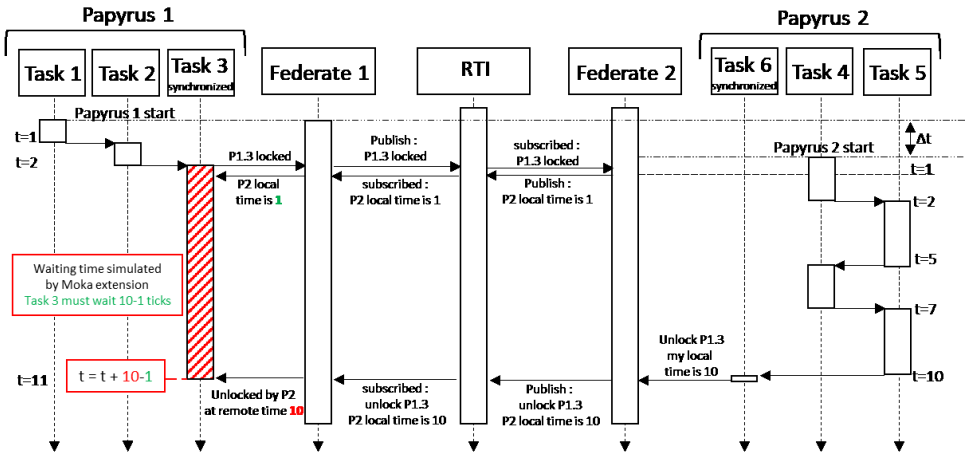

Figure 6 : Sequence flow diagram of two Papyrus with synchronized point

At the beginning of the sequence flow diagram, each Papyrus instance has its own local time and start its local process. To make it more realistic, each Papyrus instance starts at a different time. Δt represents the time gap between two Papyrus instances launches. It impacts time synchronization in order to illustrate time management mechanism proposed in this contribution. When P1 is locked (at t=2), Papyrus 1 federate (F1) publishes this information to all other federates. Then, the federate which will unlock P1 in the future must send back its own local time. Here, when P1 is locked, P2 is at t=1. P1 is now waiting to be unlocked and its local time is frozen.

P2 process execution continues until it has to unlock P1.6. The federate publishes this information with its own local time t=10. Federate 1 receives this information and knows which federate unlocked it. It can now determinate the duration of the lock: duration of P1.3 synchronized task = Tend - Tbegin = 10 - 1 = 9 ticks. Now, the Moka extension can unlock its local time and create a waiting time with a duration of 9 ticks. This process allows the use HLA mechanism in order to synchronize the local time of the different Papyrus instances.

A more complex example was setup in order to test synchronization mechanisms. As a result, figure 7 is a BPMN diagram describing this simulation scenario. P1 starts and unlocks P2 & P3. Secondly, P1 must wait for P2 and P3 to finish their own simulation. Another interesting point is that P3 releases P1 after P2, even if P1 must first wait for P2. In that case, P1 stays naturally locked and wait for P2 message. Then, P1.5 is immediately unlocked (more explanation later in the paper with Figure 9).

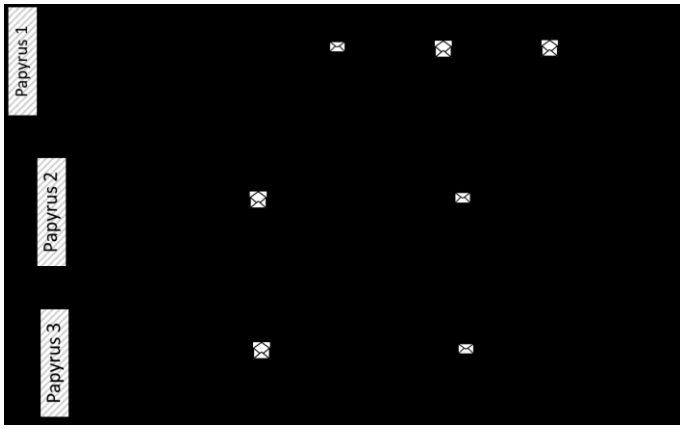

Figure 7 : BPMN process of 3 interconnected Papyrus

Figure 8 describes the technical architecture of our example. As introduced in Figure 7, there are three Papyrus instances working together. Each one of them has its own Hybrid HLA/FMI federate allowing communication between them through HLA Run Time Infrastructure.

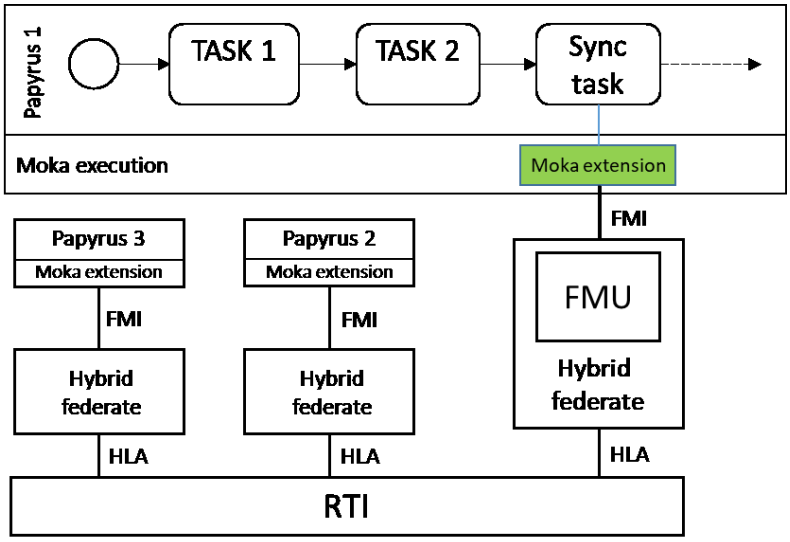

Figure 8 : Logical architecture of Papyrus Distributed Simulation

Figure 9 illustrates a second sequence flow diagram extracted from our example. To simplify graphical rendering, we decided to merge all local tasks and each instance of Papyrus in one single line: "Local tasks". We also merged the synchronized task with the hybrid federate line (excepted for Papyrus instance 1), and hide the RTI component.

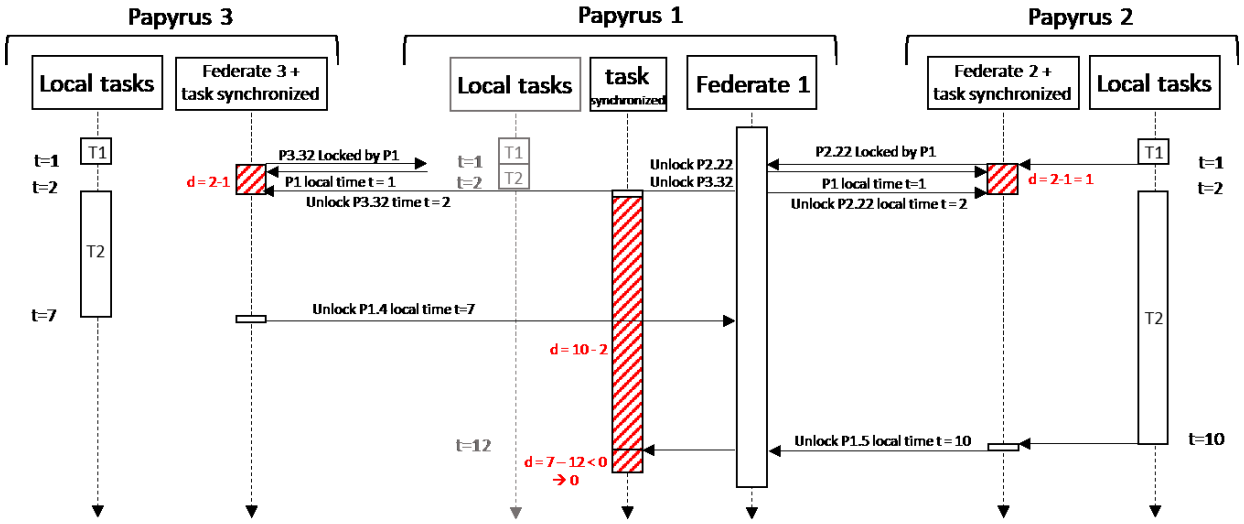

Figure 9 : Sequence flow diagram of Figure 5 BPMN Process

According to Figure 9, we can track execution steps of Figure 7 model. This example permits to test delay algorithm with a specific study case. Indeed, P1 must be firstly unlocked by P2.24 and secondly by P3.34. When P1 enters in P1.3 lock, it asks to P2 about its current local time: starting time = 2 and stores this information for later.

P3 process is faster than P2. As a result, Figure 9 diagram shows that P3 send a release message at t=7, but P1 is not yet locked by P3. So Federate 1 save this message and does not impact P1 simulation.

Later, at t=10, P2 releases P1.4 lock. At this time, the federate first unlocks P3.34 and determines its delay: d = release time (10) - starting time (2). Moka engine of Papyrus instance 1 can unlock its local time and set delay time to 8.

Then subsequently, the execution process goes in a second synchronized task P1.5, but it is immediately released: d = release time (7) - starting time (12) < 0 so no delay. Then, the distributed simulation ends since each Papyrus instance has reached its final task.

5 CONCLUSION

Papyrus community tends, in upcoming versions, to open the software to co-simulation with the implementation of the FMI standard. Nevertheless, as FMI does not include time management. To overcome this limit, we have presented in this work, a Moka federate component extended to communicate with an HLA RTI. This first result is pushing the platform capabilities to new time orchestrated features. It will allow developers to implement synchronization points between Papyrus instances.

The distributed simulation is currently at the early stage of design and implementation. So far, it can only share simple tokens as resources. Much remains to be done such as implementation of performance evaluation. For instance, requesting an FMU file and exchanging messages through a federation is time consuming compared to a standard HLA based DS compliant tool. Another interesting future work would be to implement more complex shared resources between Papyrus instances. To reach this goal, we plan to involve semaphore mechanisms. Finally, since another goal of the work presented in this paper is dealing with interoperability at large (e.g. databases, matlab files), we trust that HLA and FMI standards will facilitate connection with other heterogeneous tools or components.

REFERENCES

Altuhhova, Olga, Raimundas Matulevicius, and Naved Ahmed. 2013. “An Extension of BPMN for Security Risk Management.” International Journal of Information System Modeling and Desigh, January, pp 93-113.

Bastian, Jens, Christoph Clauß, Susann Wolf, and Peter Schneider. 2011. “Master for Co-Simulation Using FMI.” Proceedings of the 8th International Modelica Conference, March, 115–20.

Better, Marco, Fred Glover, Garry Kochenberger, and Haibo Wang. 2008. “Simulation Optimization Applications in Risk Management.” International Journal of Information Technology & Decision Making, Vol. 7, No. 4, 2008, World Scientific Publishing Company edition.

Blochwitz, Torsten. 2016. “Functional Mock-up Interface for Model Exchange and Co-Simulation.” https://www.fmi-standard.org/Downloads.

Bouanan, Youssef, Simon Gorecki, Judicael Ribault, Gregory Zacharewicz, and Nicolas Perry. 2018. “Including in HLA Federation Functional Mockup Units for Supporting Interoperability and Reusability in Distributed Simulation.” Proceedings of the 50th Computer Simulation Conference, July 2018.

Ellner, Ralf, Samir Al-Hilank, Hohannes Drexler, Martin Jung, Detlef Kips, and Michael Philippsen. 2011. “A FUML-Based Distributed Execution Machine for Enacting Software Process Models.” ECMFA : Modeling Foundations and Application 2011, 2011.

Franke, Rûdiger, Sven Erik Mattsson, Martin Otter, Karl Wernersson, Hans Olsson, Lennart Ochel, and Torsten Blochwitz. 2017. "Discrete-Time Models for Control Applications with FMI." In , 507–15. https://doi.org/10.3384/ecp17132507.

Gorecki, Simon, Youssef Bouanan, Judicael Ribault, Gregory Zacharewicz, and Nicolas Perry. 2018. "Including Co-Simulation in Modeling and Simulation Tool for Supporting Risk Management in Industrial Context." International Multidisciplinary Modeling & Simulation Multiconference, September 2018.

Guermazi, Sahar, Jérémie Tatibouet, Arnaud Cuccuru, Saadia Dhouib, and Sébastien Gérard. 2015. "Executable Modeling with FUML and Alf in Papyrus: Tooling and Experiments." EXE MoDELS, 2015.

IEEE Computer Society. 2010a. 1516.2-2010 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-- Object Model Template (OMT) Specification.

IEEE Computer Society. 2010b. 1516-2010 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-- Framework and Rules.

IEEE Computer Society. 2010c. IEEE Standard 1516-2010 for M&S - HLA - Framework and Rules. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5553440.

OMG Publication. 2018. Semantics of a Foundational Subset for Executable UML Models (FUML). https://www.omg.org/spec/FUML.

Sievert, Nicke. 2016. "Modelica Models in a Distributed Environment Using FMI and HLA."

Yilmaz, Faruk, Umut Durak, Koray Taylan, and Halit Oguztuzun. 2014. "Adapting Functional Mockup Units for HLA-Compliant Distributed Simulation." 10th International Modelica Conference, March 2014.

AUTHOR BIOGRAPHIES

SIMON GORECKI is MSc and Ph.D. Student at University of Bordeaux in IMS Lab. His research domain is about simulating process with distributed simulations and HLA (High Level Architecture). His email address is simon.gorecki@u-bordeaux.fr

JUDICAL RIBAULT is Ph.D. and part time Researcher at University of Bordeaux and IMS Lab. His research interests are methodologies for modeling and simulation.

GREGORY ZACHAREWICZ is Full Professor at IMT – Mines Ales and LG2IP Lab with both competences in enterprise engineering, discrete event modeling & distributed simulation. His email address is gregory.zacharewicz@u-bordeaux.fr

NICOLAS PERRY is Full Professor at ParisTech ENSAM of Bordeaux. Hi domain of research is about system engineering, product process integration and green manufacturing. His email address is nicolas.perry@ensam.eu

YVES DUCQ is Full Professor at University of Bordeaux in IML Lab. His domain of research is about enterprise modeling and process performances .His email address is yves.ducq@u-bordeaux.fr