Project Number: 257909

SPRINT

Software Platform for Integration of Engineering and Things

Collaborative project
Information and Communication Technologies

D4.17 Real time simulation service definition

Start date of project: October, 1st 2010
Duration: 41 months

Deliverable	Real time simulation service definition		
Confidentiality	PU	**Deliverable type**	R
Project	SPRINT	**Date**	2014-02-07
Status	Final	**Version**	1.0
Contact Person	Otto Tronarp	**Organisation**	WOLFRAM MathCore AB
Phone	+46 13 328 500	**E-Mail**	ottot@wolfram.com

THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE SVENTH FRAMEWORK PROGRAMME UNDER GRANT AGREEMENT N° 257909

AUTHORS TABLE

Name	Company	E-Mail
Otto Tronarp	WOLFRAM MathCore AB	ottot@wolfram.com
Stefano Boccabella	ALES S.r.l.	stefano.boccabella@ales.eu.com
Massimiliano D'Angelo	ALES S.r.l.	massimiliano.dangelo@ales.eu.com
Alberto Ferrari	ALES S.r.l.	alberto.ferrari@ales.eu.com
Alessandro Mignogna	ALES S.r.l.	alessandro.mignogna@ales.eu.com
Christos Sofronis	ALES S.r.l.	christos.sofronis@ales.eu.com

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected

CONTENT

Content of Figures

Content of Tables

1 Introduction

1.1 Overview, Purpose and Scope

The purpose of this document is to describe the design of the real time simulation service in the Sprint Engineering Environment (SEE).

The role of the real time simulation service within the SEE is to enable early verification of the integration of the system components collaboratively designed in SEE. The design of the system components (Figure 1-1) occurs in a distributed fashion. At each site, the design starts from a set of requirements, which might be captured in the form of contracts. In the first phase of the design process, models are defined as virtual prototypes of the system components. Later, physical prototypes may become available. Testing the integration of the system components during the design process must hence face two major challenges:

- The system component prototypes, models and physical devices, are distributed in different locations
- The system component prototypes are heterogeneous; the integration must combine together models and physical devices. Moreover, models may be defined using different modeling tools.

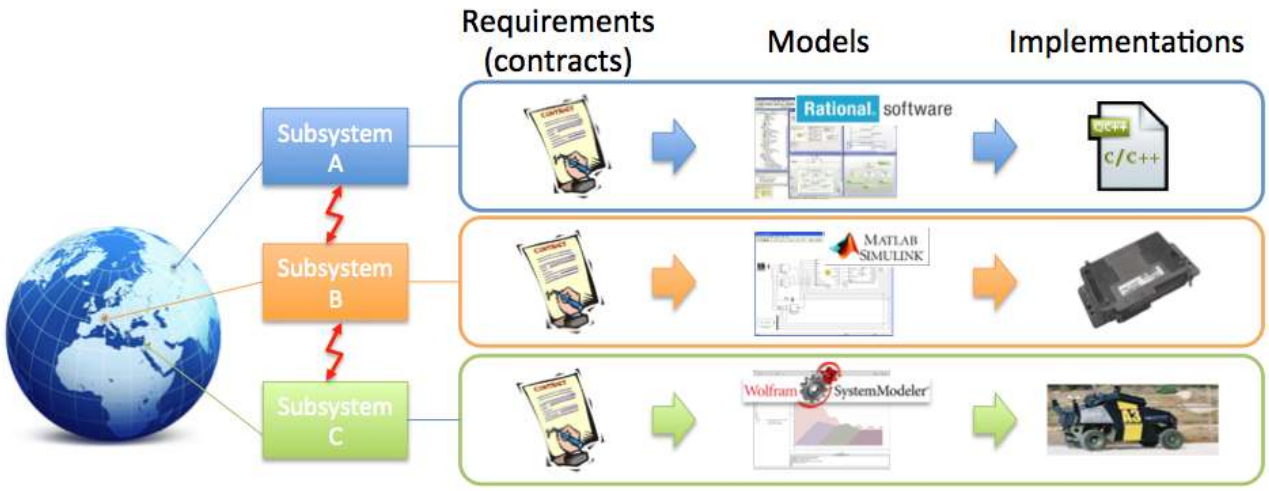

Figure 1-1: Distributed design process

There are a number of reasons why distributed simulation is a requirement in such scenario:

- Partners might not be willing to share models with each other for IP reasons. If a system with sub-systems modeled by different partners are to be simulated it must be done distributed with each sub-system simulation running at their respective partner's site.
- Physical devices cannot be easily shared. Due to the size of the component, it might be difficult to ship a device in a certain location to perform testing. Moreover, shipment would introduce extra delays in the design process. Last, the availability of physical devices might be limited; solving the integration issue by shipping the prototype would prevent the usage of the prototype during the testing phase. Distributed simulation, instead, would not only enable the integration while maintaining the device at the original site, but also make it available for other design activities.
- All partners might not have access to or the possibility to use all the tools used by other partners in the project due to for example licensing costs, lack of know-how, etc. This case can also be solved by employing distributed simulation, but it can also be solved by using hosted simulation. For that to work all participating tools needs to be able to export to a common executable model exchange format and each partner needs to have access to one tool that can perform hosted simulation using that format.

The purpose of the real time simulation service is to provide a solution to these obstacles to the system engineering process. SPRINT requirement 6.1.8 in [15] states "The SPRINT platform shall adopt standard or

de-facto standard technologies wherever available". Therefore, the proposed solution builds on top of existing and emerging standards like High Level Architecture and Functional Mock-up Interface [30].

High Level Architecture (HLA) has been an IEEE standard (IEEE1516) since 2000. It provides an architecture for coordination and execution of distributed simulations in real time. In the SPRINT real time simulation service HLA provides the foundation for distributed real time simulations. Making it possible to integrate different simulators into the simulation. These simulators can be stand alone simulators or simulators that hosts the execution of virtual model implementations in the form of FMUs. They can also act as proxies for physical devices, making it possible to integrate physical devices into the simulation.

Functional Mock-up Interface (FMI) is a rising standard developed under the umbrella of the Modelica Association, there are currently around 50 tools that supports or plans to support the FMI standard [13]. It provides an open and standardized C interface to execute models. The role of the FMI in the SPRINT real time simulation service is to provide a way to exchange executable model implementations between different modelling tools in the SPRINT tool chain to be executed by a FMI enabled simulator.

The SPRINT real time simulation service provides the middle ware that makes it possible to take a system model, from any modelling tool integrated into SPRINT, and easily deploy it as a distributed real time simulation. In the deployed simulation different system components may come from different modelling tools or they may be actual physical prototypes of component.

2 State-of-the-art and innovative contribution

The real-time simulation service platform merges together three different simulation techniques

- The distributed co-simulation, i.e. the execution of a simulation with the modelling components executed in different simulators running in a coordinated and distributed fashion
- The HiL simulation (or Real-time distributed simulation), which involves both physical devices and simulators and which must guarantee the coordination under the real-time synchronization constraint
- The hosted simulation, which allows executing models defined in different tools in a single hosting tool, by defining common model executable formats and execution protocols.

In this section, we will present the state-of-art for each of these areas, together with the relevant standards. Last, we will point out the innovative contribution of the SPRINT simulation platform.

2.1 Distributed co-simulation

Distributed co-simulation involves the execution of different parts of a system model over different simulators. The set of simulators can be heterogeneous and eventually distributed in different locations. The integration of different simulators allows executing different kind of models. Simulators must cooperate exchanging data produced by interconnected models and synchronizing each other. This mechanism impacts the performance of the simulation, which may become extremely slow when the interaction among the simulators occurs over a communication medium with large delays.

Several works exist in literature on distributed simulation. A middleware for distributed simulation based on RESTful Web-Services is presented in [1]. It is implemented on top of the CD++ engine, a simulator based on the Discrete Event System Specification (DEVS) formalism. The architecture is based on a main server that the user communicates with and supporting servers that are used as simulation slaves. It includes general simulation management services for uploading models to the main server, starting/stopping simulations and storage of simulation artifacts. The main server takes care of replicating the necessary parts of the model to its supporting servers for simulation. Participating simulators communicate with each other by exchanging XML messages through the REST interface.

A SOAP based distributed simulation environment built on top of SystemC is presented in [2]. Here each participating SystemC IP is wrapped with an automatically generated wrapper that provides a SOAP interface to the IP as well as a synchronization mechanism. At startup each IP register its interface at a web registry service using WSDL. A client can then setup up a simulation by discovering distant IPs through the web registry service.
The EU funded project DIESIS (Design of an Interoperable European federated Simulation network for critical infrastructure) has developed an architecture for distributed simulation that is presented in [3]. Here they abandon the HLA approach of generic simulator coupling and instead employ a scenario oriented design. Simulators are made pair-wise or group-wise compatible using arbitrary coupling primitives if and only if that coupling is required for the chosen scenario. Thus there is no need for a central federate object model like in HLA. The simulator coupling is done using the DIESIS interoperability middleware which is based on SOAP. The middleware also includes several support components, the most fundamental ones are FCM, TMM and KBS. The federation control module (FCM) provides the centralized control for managing the simulation. The time management module (TMM) provides event based time synchronization services for the participating simulations. The knowledge base system (KBS) holds ontological representation of the domains and their relationships. Participating simulators can use that information to automatically generate the necessary data links for the simulation.
Despite the large attention that distributed simulation has received in literature, its application in the industry in general is rare. One domain that has a strong adoption of distributed simulations is the military domain with High Level Architecture (HLA) starting out as a standard for US military simulations in the late 1990's. Since then HLA has spread to also being heavily used in NATO countries as well as Partnership for Peace countries. Another domain where distributed simulation has large application is the gaming industry, for the creation of distributed multi-player on-line gaming platforms. In [4] they argue that the lack of use of distributed simulation in general, and HLA in particular, within other industries is due to the lack of support in COTS simulation software. Furthermore, their survey indicates that when the distributed simulation is applied by an organization it is more common to do it by using an in-house architecture based on low level technical standards than to actually use a standard architecture for distributed simulation.

The survey done in [5] to study future trends in distributed simulation also indicates that the industry adoption of distributed simulation in general is low, with the exception of the military sector. They attribute it in part to

the lack of plug-and-play solutions for industrial usage. However, their survey shows a growing interest for distributed simulation technologies in the high-tech industry driven by the globalization of the development and manufacturing processes. Our scan of the literature on distributed simulation has highlighted the usage of different technologies to implement the coordination among the simulators. OMG Corba, OMG Data Distribution System (DDS), High Level Architecture and, lately, Web services, are the most common choices. Corba, DDS and Web services simply provide different ways to abstract the communication among remote entities, and are not specifically targeted for simulation. HLA, instead, provides a full set of services specifically designed for distributed simulation, which serve both the communication and the synchronization among the simulators. A description of the main features of HLA follows.

2.1.1 HLA

High Level Architecture (HLA) was originally developed by the US DoD in the mid 1990's as an effort to get control of the spiralling costs of developing dedicated simulators for each new problem and multiple organizations crating simulations for similar systems [6]. It eventually became the prescribed standard for military simulation within the US and from there it has spread as a standard for simulation within NATO countries and even Partnership for Peace countries. In 2000 HLA was established as an IEEE standard (IEEE1516).

The usage of HLA outside of the military domain is scarce [4]. One of the few industries where HLA has taken off is within the space industry; the European Space Agency for example is even developing its own HLA implementation named EODiSP [7]. In [4] they argue that one of the reasons for the lack of use of HLA within other industries is the lack of support for HLA in COTS simulation software. Nowadays there is support for HLA available in some COTS software, for example the HLA Toolbox and HLA Blockset for MATLAB/Simulink from FORWARDSIM.

From an architectural standpoint, a distributed simulation using HLA is composed by a set of simulators coordinated by a Run-time Infrastructure (RTI) module (Figure 2-1). The set of simulators participating in a distributed simulation forms a federation, and each simulator is a federate.

The RTI provides a set of services that allows the simulators to cooperate. In particular, the RTI enables:

- Communication among the models running in different federates
- Synchronization of the federates

An overview of the HLA synchronization and communication mechanisms will be provided below, so as to introduce the reader to the main concepts that will be used in the next sections. We refer the reader to [6] for a more complete description.

Communication among the models running in different federates

The execution of a system model in a distributed fashion requires to guarantee the communication among the modeling components that are interconnected in the original model but have been distributed for execution on different federates. Communication among the models in HLA can be performed using object classes and interactions.

An object class of a particular type is characterized by a set of attributes, and it is typically used to describe things that can persist. A federate can instantiate object classes of a particular type, and take care of updating the attributes of the object class instances that it owns. Another federate can subscribe to an object class of a particular type; it will discover new instances of the object classes that are created, and it will receive all the updates of the attributes of these object instances.

An interaction class of a particular type is characterized by a set of parameters, and it is typically used to describe things that do not persist, like events. A federate can send interactions of a particular type. Another federate that is subscribed to the same interaction type will receive all the interactions of that type.

Messages exchanged among the federates, in the form of updates of object classes or of interactions, can be delivered in two ways: Time Stamp Order and Receive Order. Time Stamp Order messages carry a time stamp, and they are delivered to the recipient federate in chronological order, when the internal time of the federate is equal to the timestamp of the message. Receive Order messages are delivered as soon as they are transmitted.

Synchronization of the federates

HLA supports both conservative and optimistic synchronization approaches. A conservative approach guarantees that a federate always receives the TSO messages in chronological order; a federate cannot advance its time to T if TSO messages with timestamp lower than T might still be received. Optimistic

synchronization, instead, allows a federate to advance its time with no guarantees on the ordering of the TSO message; if a TSO message arrives with a timestamp in the past with respect to the current time of the federate, the federate is supposed to perform a roll-back so as to schedule the message at the proper time.

In both cases, the synchronization is based on the explicit request by a federate to the RTI of advancing its time. To grant the advancement, the RTI considers the internal time and the synchronization approach adopted by the federate issuing the request and by all the other federates.

Synchronization can be time-stepped or event-based. In the time-stepped approach, a simulator requires advancing to a certain time T; when the time advance is granted, the simulator moves to the requested time. In the event-based approach, the simulator requires to advance to the next event (TSO message) notified by the other federates or to a time T. When the time advance is granted, the simulator moves to the time of the next event, if an event is available before time T, or to time T.

Notice that all federates participating in a distributed simulation do not need to agree on a common synchronization approach; the synchronization approach adopted by a federate (conservative vs optimistic, time stepped vs event based) is completely transparent to the other federates.

A federate may play one of the following roles in the synchronization of the federation: time regulating, time constrained, both time constrained and time regulating, neither time constrained nor time regulating. A time regulating federate serves as a reference for the synchronization of the other federates. The time constrained federates adjust their time advancement according to the time regulating federates. Time regulating federates are hence always granted to advance their time, whereas time constrained federates must advance their time according to the status of the time regulating federates. Time regulating federates can send TSO messages, but they can only receive RO messages; since they progress their internal time independently from the other federates, it would be impossible to guarantee the TSO reception. Time constrained federates, instead, can receive messages in TSO order, but they cannot send TSO messages.

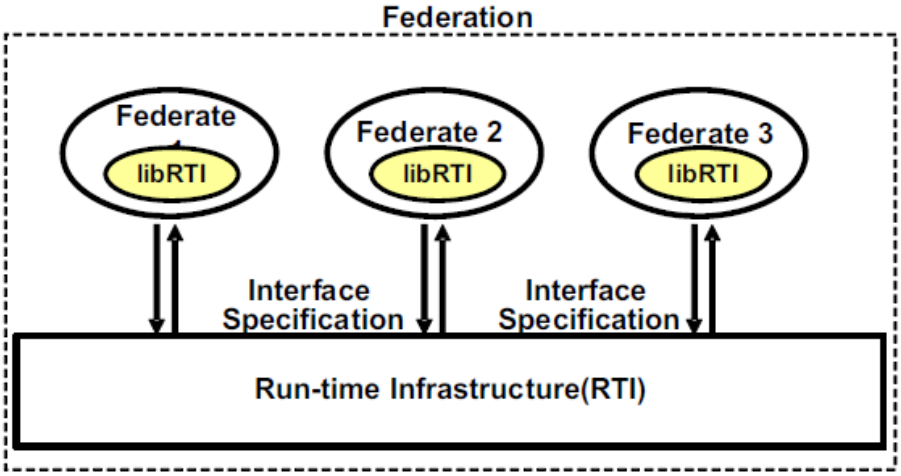

Figure 2-1: A HLA simulation structure

2.2 Distributed Real-Time Simulation

Hardware-in-the-loop (HIL) is a technique in which a portion of a given system is a simulation algorithm and a portion of the same system is a hardware implementation. The simulated and the physical part need to communicate to exchange data and stay synchronized. Since physical devices cannot (in general) be suspended, the whole model execution occurs in real-time, and any delay external to the simulated model (due to the communication between the simulated and the physical part, or to the simulated model execution) may affect the simulation results. The introduction of physical devices in the distributed simulation scenario hence imposes a real-time constraint on the execution. The usage of distributed simulation to build virtual environments for training of soldiers (typical application in the military domain) or for gaming still requires a real-time execution of the simulation, with the purpose of making the interaction with the user more realistic. However, when the real-time constraint cannot be met, the whole execution can eventually slow down affecting just the user experience. In the scenario with physical devices, instead, violating the real-time constraint may affect the correctness of the simulation trace.

In HiL simulation, the real hardware is usually connected via a digital interface to a computer-simulated system model, in an ad-hoc way. Real-time operating systems may be used to guarantee deterministic delays in the simulated model execution. Guarantee the synchronization of simulators and physical devices over the Internet is more challenging; distributed real-time simulation is hence not very common. Publications on this topic are appearing in literature nowadays [8].

To the authors' knowledge, standards for distributed real-time simulation do not exist. However, the OPC foundation has defined a standard interface, described in the next section, for interoperability with physical devices, which might be exploited for the interaction between the simulators and the physical devices.

2.2.1 OPC / OPC UA

The OPC series of standards is maintained by the OPC Foundation. The name comes from the original standard developed in 1996, OLE for Process Control where the name indicated that the specification was built on Microsoft's OLE, COM and DCOM technologies. It defined a standard for data access to industrial automation systems from Windows based computers. Since then the standard has evolved into several specifications for communication of real time data, historical data, alarms and events between software systems, sensors, instruments and controllers. The standard is widely adopted in the industrial automation and process industry with over 2500 vendors supplying over 15000 OPC enabled products [9].
The COM/DCOM solution suffers from several problems ranging from problems with firewalls to the fact that it's bound to the Windows platform. When Microsoft announced that they would discontinue the OLE/COM/DCOM technology in 2002 it was the tipping point and the OPC Foundation started the work on a new standard [10]. This work evolved into OPC Unified Architecture (UA), a standard based in part on web technology. The OPC UA communication stack is shown in Figure 2-2 and as can be seen it utilizes web technologies like XML/SOAP/HTTP but as a parallel path for performance reasons they have defined custom binary protocols as well. OPC UA defines services for real time data access, historical data access and alarms.
OPC UA could fill a role in SPRINT as a protocol for integration of physical devices and since it is based on web technology it really fits well with the spirit of SPRINT and Internet of Things in general. But OPC UA is not an open standard; you have to be a member of the OPC Foundation to get access to it. That and the lack of freely available implementations lead to the decision to not include OPC UA in the architecture of the SPRINT real time simulation service.

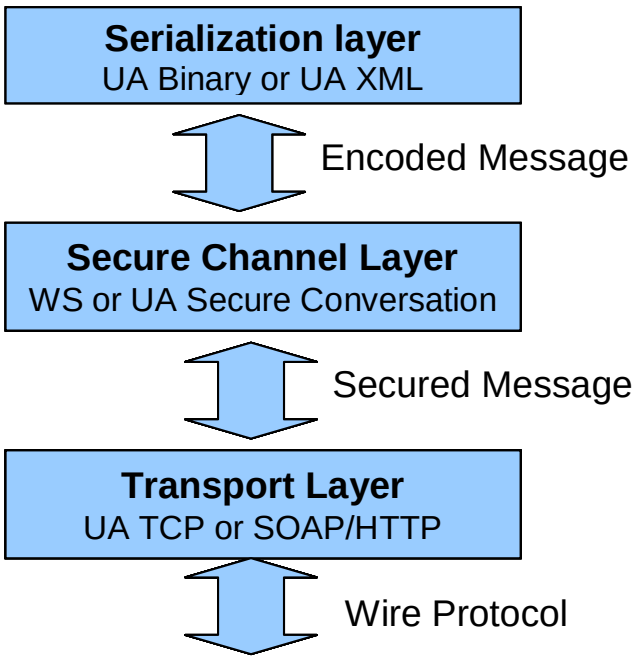

Figure 2-2: The OPC UA communication stack

2.3 Hosted simulation

The hosted simulation technique enables the joint execution of models defined using different modelling tools. While co-simulation addresses this issue combining together different simulation tools, each one capable of running different kind of models, hosted simulation executes all the models in a single tool (hosting tool). The key enablers are

- a common executable model format to which all the modelling tools must export their own models
- a hosted simulation protocol which must be implemented by the hosting tool, so as to properly execute the models.

With respect to co-simulation, hosted simulation may achieve better performance because there is no need for coordination among different tools.
The main relevant specification on hosted simulation are the Functional Mock-up Interface (FMI) and the SPEEDS Hosted simulation. Their description follows.

2.3.1 FMI

Functional Mock-up Interface for Model Exchange and Co-Simulation was initiated and organized by Daimler AG within the EU funded MODELISAR project. Version 1.0 of the FMI standard was released 2010 and version 2.0 is planned for released in 2012 [11]. The MODELISAR project ended in 2011 and the maintenance and further development of the standard is about to be taken over by the Modelica Association [12]. For being a young standard it has quite wide adoption, according to [13] there are around 50 different tools that either supports or plans to support the FMI 1.0 standard [30].
FMI defines a C interface that is implemented by an executable called FMU (Functional Mock-up Unit). A simulation environment can use the FMI interface to create one or more instances of the FMU and simulate them together with other FMU's or models native to the simulation environment. There are two versions of the standard, FMI for Model Exchange and FMI for Co-Simulation.
The FMI for Model Exchange defines an interface to evaluate the equations of a dynamic system that is described by differential, algebraic and discrete-time equations. The available data flows in FMI for Model Exchange is shown in Figure 2-3.

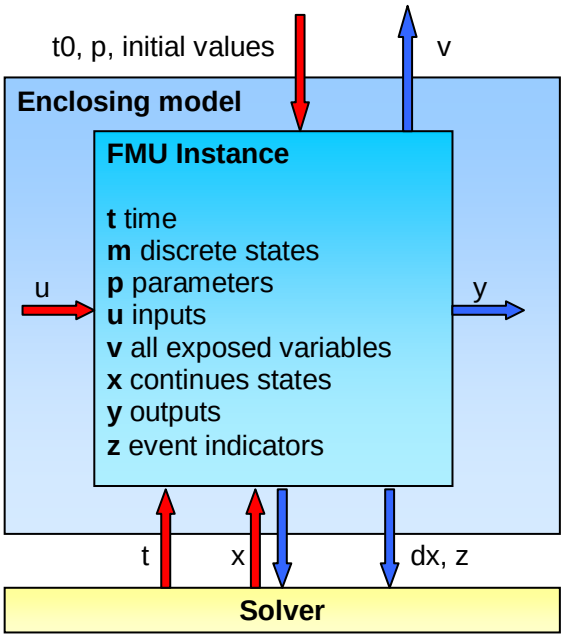

Figure 2-3: Data flow between an FMU for model exchange, the enclosing model and solver. Blue arrows are information provided from the FMU and red arrows are information provided to the FMU.

The FMI for Co-Simulation defines an interface for advancing a time dependent system in time, the FMU in this case is either self-contained and has a bundled solver or a communication layer for communicating with its native simulation environment where the actual integration takes place. The communication between the FMU and the co-simulation master takes place at discrete communication points; the data flow is shown in Figure 2-4.

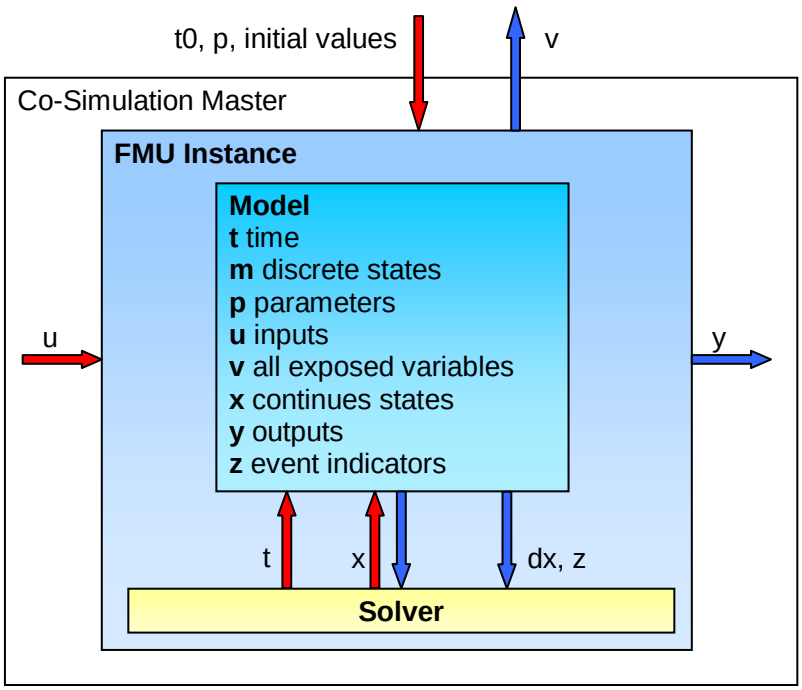

Figure 2-4: Data flow between an FMU for Co-Simulation the co-simulation master. Blue arrows are information provided from the FMU and red arrows are information provided to the FMU.

An FMU is distributed as a zip file containing:

- FMI Description file (XML)
- The C sources for the FMU including runtime dependencies and/or binaries for one or several target machines.
- Additional resources in file format specific to the tool that created the FMU.

According to the authors knowledge, standards for HiL simulation do not exist. However, the OPC foundation has defined a standard for interoperability with physical devices, which might fit in the context.

2.3.2 SPEEDS Hosted Simulation

SPEEDS Hosted Simulation was developed as part of the SPEEDS project [26] and enables the execution of a system model composed by Heterogeneous Rich Components (HRC) designed in different tools. An HRC component is exchanged in the form of an XMI description of the component and an implementation, compliant with the SPEEDS API, either as source files or as a binary library + header files.

2.4 Innovative contribution

The main innovative contribution in the definition of the SPRINT Real Time Simulation service platform is the combination of the previously described simulation techniques with a set of services for resource sharing provided by the SSI/SII. The resulting infrastructure largely improves the usability of the distributed simulation. The user can browse “the cloud” to discover the available resources for simulation, namely models, physical device and simulators, and combine them to specify the simulation to be performed (which models and physical device must be used, how they must be deployed on the available simulators). At the end of the simulation, the results generated by the distributed simulation entities are shared, and become another resource in the cloud that the user can access.

The Real Time Simulation service platform will hence rely on a set of resources providing services that enable (but must not be limited to) the setup and the management of a simulation. Resources can be accessed to inspect their status, before, during and after the simulation. A user might control the CPU load of the servers running a particular simulator, so as to deploy the model execution where more computational resources are available; visualize in real-time a probe in a model during the simulation execution; access a physical device to verify its availability before starting the simulation.

The candidate technologies for the SSI/SII infrastructure allow establishing semantic links among the different resources. Simulation results can be linked together, and put in relation with the simulation configuration they have been generated from, which in turn combines models, devices and simulators.

Notice that the standards for (real-time) distributed simulation (e.g., HLA) are focused on the execution of the

simulation, namely run-time coordination and communication among the modeling components, and do not address the management of the “things” involved in the simulation.
Another research topic is the definition of methods for effective (real-time) simulation over the Internet. The communication link does not provide, in general, any guarantee on the maximum delay and jitter. A correct composition of the executions of the distributed simulation actors may require to investigate new HiL simulation approaches; techniques to decouple to a certain extent the execution of simulators and physical devices may guarantee some parallelism, and relax the constraints on synchronization (e.g., buffering of time-stamped data generated by the devices). Moreover, the theory on stuttering invariance [14] will be exploited to identify classes of devices whose behavior is insensitive to delays, and can hence be integrated across the Internet without suffering from the extra communication latency. Besides the issue on the correct composition of the executions, the impact on simulation performance of the communication delays might also need to be mitigated; internet delays put an overhead on the communication/coordination among the simulation actors, which might result in slow simulations.

As a last innovation point, the real time simulation service platform will support the contract-based design methodology, and the peculiarities of the platform will enable new monitoring scenarios. The distributed infrastructure supports the execution of monitors and of the monitored components in different locations (remote and distributed monitoring). Moreover, models and physical devices will expose in the simulation platform a common abstraction, which will enable the contract verification on both the virtual and physical implementations of a component.

3 Requirements and use cases

All the SPRINT requirements are described in [15]; we have collected in Table 3-1 the requirements that may have an impact on the distributed simulation service.

Req. ID	Description
6.1.8	The SPRINT platform shall adopt standard or de-facto standard technologies wherever available.
6.2.1	The communication among the components of the SPRINT platform shall be based on RESTful web services.
6.3.2	Simulation tools in SPRINT shall support interleaving semantics for the components execution.
6.3.5	SPRINT tools shall communicate via the internet with: (1) Other SPRINT tools (2) SPRINT Physical devices.
6.5.3	The SPRINT platform shall allow for the verification of contract satisfaction by using contract monitoring of deployed physical devices.
6.5.2	The SPRINT platform shall allow for the verification of contract satisfaction by using simulation and contract monitoring.
6.5.5	The SPRINT ontology shall cover the domain of testing and validation.
7.1.1	SEE shall enable the creation of Distributed Model Integration Laboratories (D-MIL), or Distributed Virtual System Integration Laboratories (DV-SIL).
7.1.2	SPRINT Engineering Environments shall allow for co-simulation of models with HiL (Physical Devices)
7.1.3	SPRINT Engineering Environments shall allow for co-simulation of models over an IP Network.
7.1.4	The communication among the components of the SPRINT platform shall be firewall friendly.
7.1.5	Distributed simulation services shall allow the integration of SystemC, Simulink, MathModellica and Rhapsody -models.
7.1.6	The SPRINT platform shall enable co-simulation of models using the following model simulators: Desyre, Rhapsody, MathModellica and Simulink.
7.1.7	The SPRINT platform shall support an existing hosted simulation technology such as FMI and SPEEDS HS.
7.1.8	The SPRINT environment should support the existing de-bugging of the co-simulation tools.
7.1.9	SPRINT simulation tools shall expose the trace information of the interfaces and externally visible state information from all components belonging to the System under Test (SUT).
7.1.10	SPRINT simulation tools shall enable the capture of trace information of selected internal variables/state information of the SUT.
8.1.1	SPRINT Engineering Environment shall allow for contract monitoring\analysis of Physical Devices.
8.1.2	SPRINT Engineering Environment shall allow for remote monitoring of Physical Devices.

Table 3-1: SPRINT Requirements related to the Distributed Simulation Service

3.1 Use cases

3.1.1 Hosted simulation

In this case the user wants to simulate a system model composed by several components, where each of the components might have been created in different tools and by different partners. The user can then use hosted simulation if the following prerequisites are fulfilled:

- Each of the participating tools can export to an executable model exchange format that the hosting simulator can import.
- Each partner is willing to share that executable with the partner that performs the simulation, i.e. there are no IP issues.

This use case is depicted in Figure 3-1. Each tool exports its model in an executable model exchange format that is used by a hosting simulator to perform the complete system simulation.

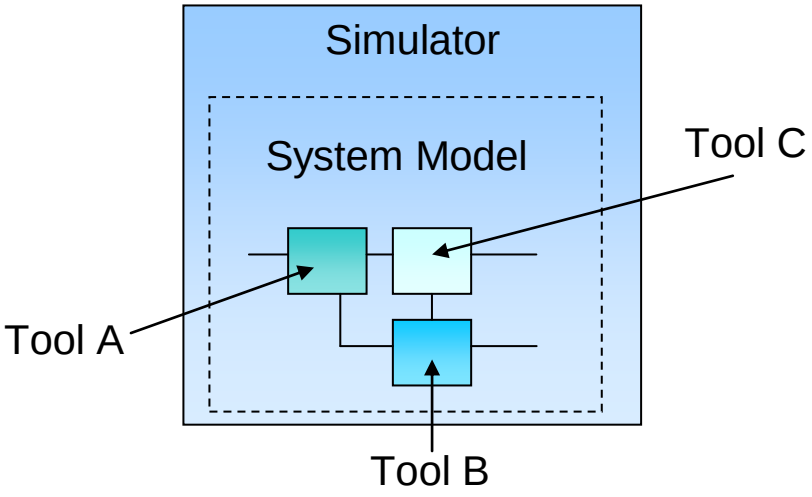

Figure 3-1: Hosted simulation.

3.1.2 Distributed simulation

Here we have the same situation as in hosted simulation, i.e. the user wants to simulate a system model composed by several components, where each of the components might have been created in different tools and by different partners. However, there are several possible reasons why the user cannot use or even does not want to use hosted simulation.

- Perhaps there are no commonly accepted executable model exchange format among the tools and the simulator.
- The complete system simulation might be too computational intensive to run on a single computer.
- All partners might not be willing to share the executable model with the partner that runs the system simulation due to IP reasons.

This use case is depicted in Figure 3-2. Here some models are executed in their native environment and others may be exported to an executable model exchange format and used in a hosting simulator. Each component or sub-system is simulated in its own simulator that can reside at different geographical locations. All of the participating simulators communicate with each other over the internet.

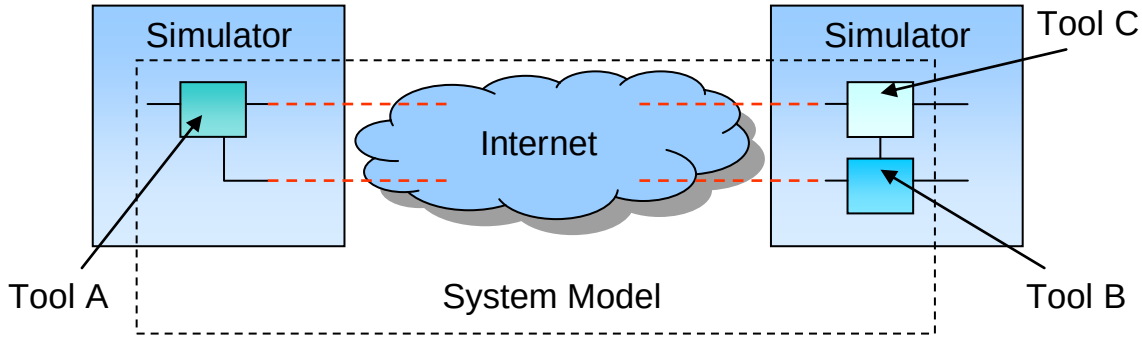

Figure 3-2: Distributed simulation.

3.1.3 Distributed (Real-Time) simulation with physical devices

In this case the user wants to simulate a system model composed by several components, where some of the components exist as physical devices and the rest only exist as models. The physical devices might for example be sensors or real implementations of the components. The main reason for doing this is to perform early integration testing. Early in the system development process the availability of the physical devices is likely to be restricted, e.g. they might only exist as early prototypes. If several partners need to perform integration testing with the same physical device, the time spent on waiting for access to the device and shipping it between partners could quickly become a bottleneck for the development process. By using distributed simulation with physical devices as shown in Figure 3-3 that bottleneck can be avoided. As in the distributed simulation use case, the components that exist as models can be simulated in their native environment or possibly in a hosting simulator. All participating simulators and physical devices communicate with each other over the internet.

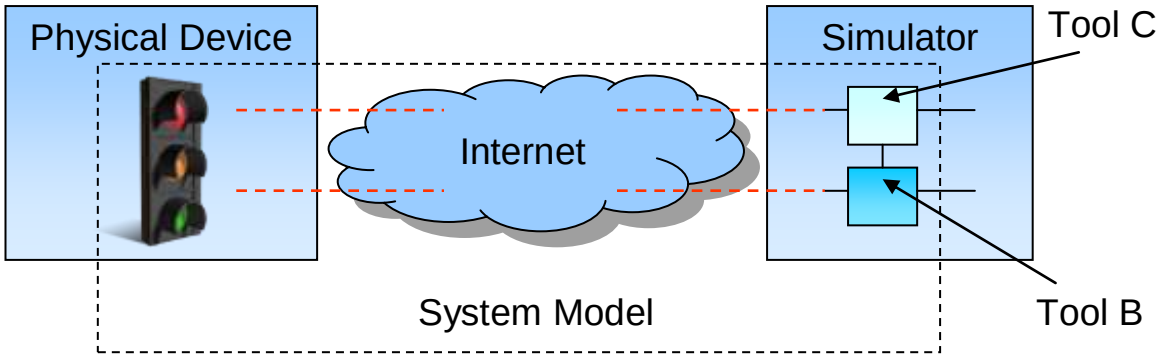

Figure 3-3: Distributed simulation with physical devices.

3.1.4 Integration testing of physical devices over the internet

In this use case all the components of the system exist as physical devices and the user wants to do integration testing of the devices over the internet as shown in Figure 3-4. All the physical devices reside at different locations and the complete system is connect over the internet.

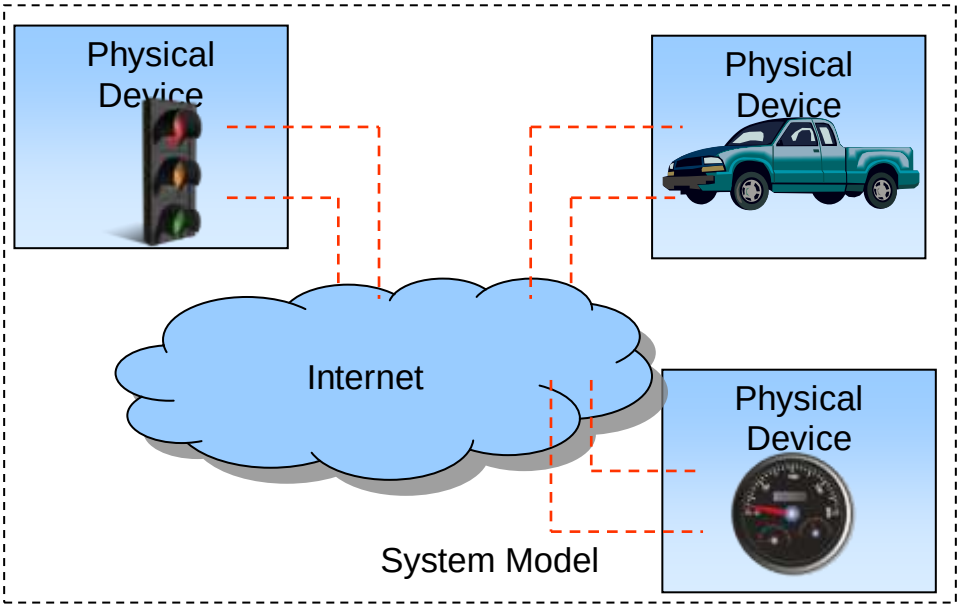

Figure 3-4: Integration testing of physical devices

4 Supported standards

Distributed simulation standards are rare in the industry and HLA is a well established standard that has seen over a decade of heavily use in primarily the military sector. SPRINT will adopt HLA as its middleware for distributed simulations. The execution of models defined in different tools will leverage in SPRINT both the co-simulation and the hosted simulation approach. The integration of simulators over HLA will provide the chance of combining the execution of different simulation tools, so as to acquire the capability to execute together models defined in different formats. At the same time, some of the tools involved in the distributed simulation may be hosting tools, capable of executing models exported by different tools in a particular model format.

The SPRINT project will adopt FMI as one of its supported model exchange formats for hosted simulation. FMI is a young standard that has quickly gained momentum and is supported by most Modelica tools as well as many other tools, either by the vendor itself or by third-party products, like for example Simulink for which it exists 3rd party solutions for import and export of FMI. Supporting FMI in the SPRINT environment would enable models from all of those tools to participate in a SPRINT simulation, avoiding the effort of integrating all the tools using HLA and with a benefit in terms of performance.

The role of the supported standards is depicted in Figure 4-1. Participating simulators have to support simulation of one or more of the hosted simulation standards or its own native models.

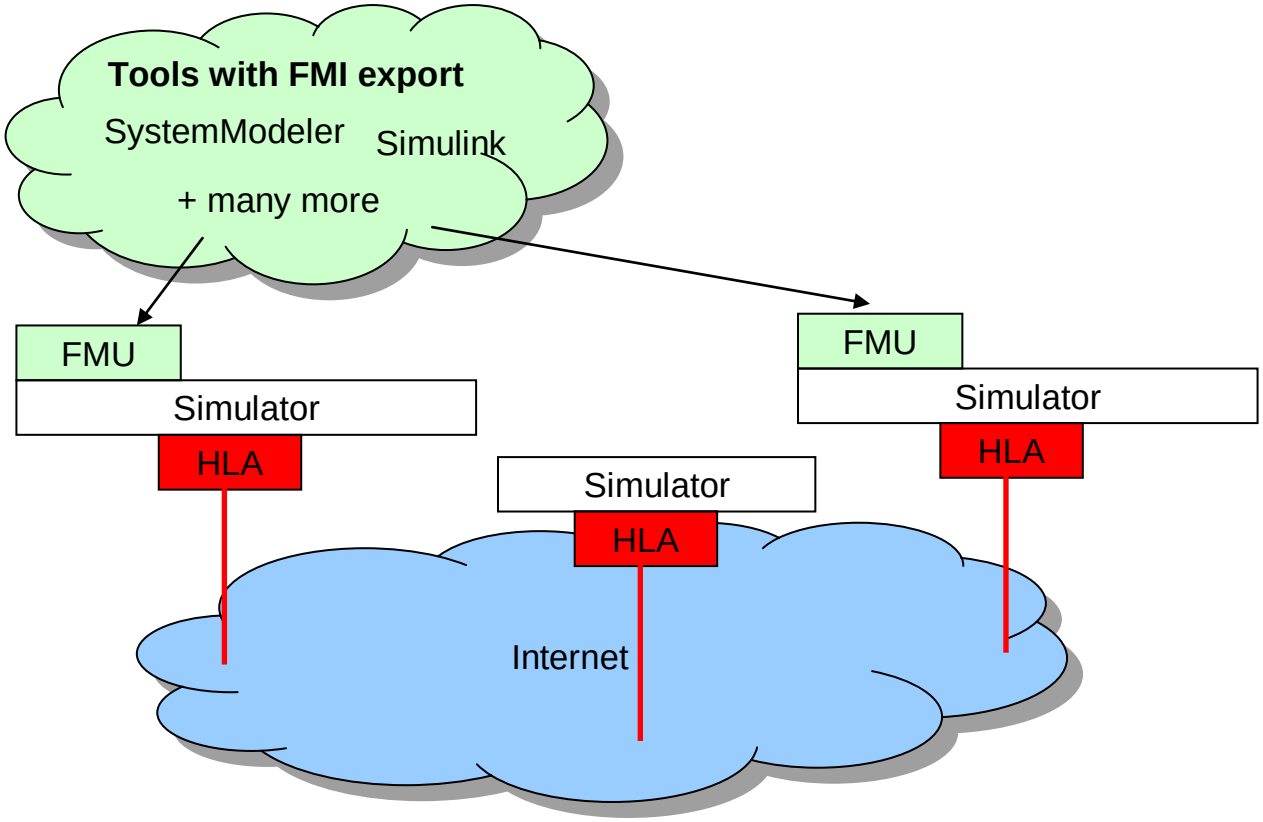

Figure 4-1: The role of the supported standards HLA and FMI in the SPRINT simulation service.

5 Simulation in the SPRINT general architecture

The general SPRINT architecture is presented in [16] and depicted in Figure 5-1. In this architecture there are two main entities that interact with the SII, Applications and Tools.

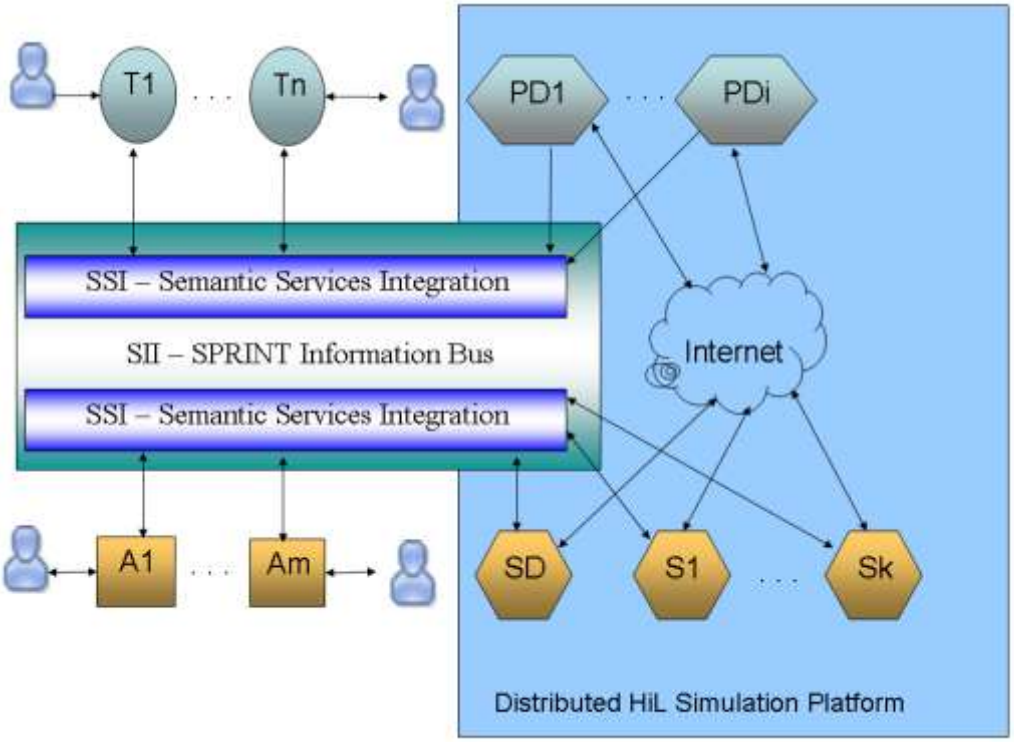

Figure 5-1: SPRINT architecture.

A tool is a software program which maintains shareable design contents, modelling some aspects of the product's design. A tool is also an independent program with its own data repository, management and usability functions that allow users to work with it, totally independent of other tools. We say that a tool holds some view of the engineered system. The content provider internal data owned by the tool is accessible to third party, may it be a tool or an application or an SII server, using REST interface.

An application is a software program which provides an added value based on the shared contents of the tools by applying functions that have not been addressed by the individual tools and which is possible due to the integration of data from multiple tools. Obviously, applications are such that they add a new value to the data in the SII repository. Notably, some tools may also play the role of an application. For instance, a design tool which can also perform analysis, simulation, and so on, and generate a report on the results of that value-adding function – is such. Application works on input data that is usually gathered through set of queries against the SII server.

The distributed HiL simulation platform consists of a Simulation Dashboard, several Simulators and physical devices (things). From the SII point of view they are a set of applications that interact with the SII. The applications also have interfaces specific for the distributed HiL simulation platform to enable coordination of the distributed simulation.

The simulation dashboard exposes the services of the distributed HiL simulation platform to the user. Through the dashboard, the user is able to specify the system model to be simulated, the simulation parameters (e.g., simulation time), control the deployment of the execution responsibilities on the simulation actors, start and control the simulation (e.g., abort the simulation, suspend the simulation), and visualize the simulation results.

Simulators acts as applications against SII, they are capable of fetching component implementations from SII and executing them. They also expose all the service interfaces required by the SPRINT platform to participate in a distributed simulation.

6 Architecture

6.1 Architectural components

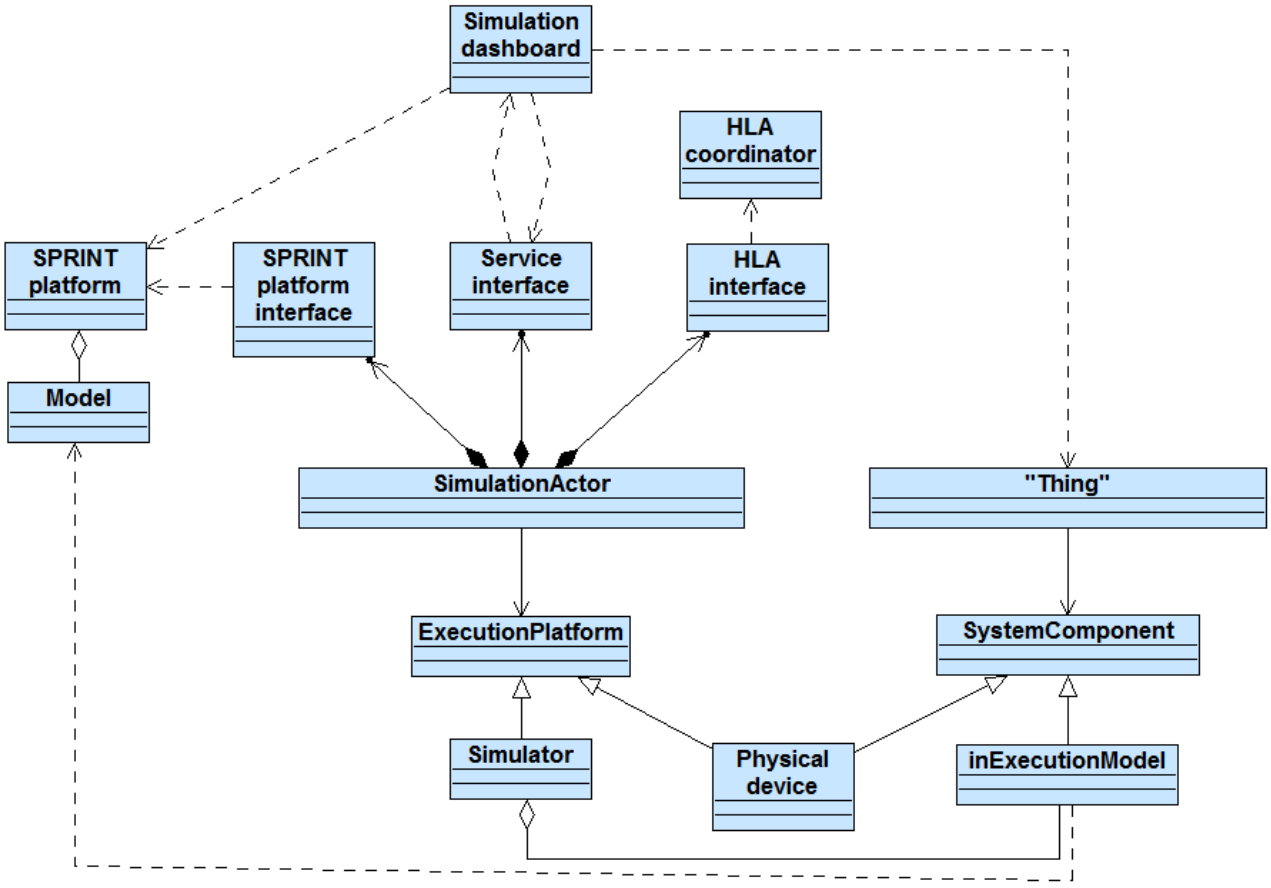

Figure 6-1: Components of the Distributed HiL Simulation platform

Figure 6-1 shows the components of the distributed HiL simulation platform and their relationships. A description of each component follows.

Simulation actor: a simulation actor is an entity that can be involved in the execution of the system prototype in the SPRINT distributed simulation platform. It acts as an adapter, exposing the service interfaces required by the SPRINT platform, and mapping them on the existing interface of a simulator or a physical device. Three kinds of interfaces are required for interaction with the other components involved in a distributed HiL simulation:

- *SPRINT platform interface* for the interaction with the SPRINT platform. The interaction concerns the registration of the provided services, the retrieval of models to be executed (just for simulators), and the publication of results generated by the simulation.
- *Service interface* must expose the simulator functionalities for access by a remote user through the simulation dashboard (e.g., starting a simulation, aborting a simulation).
- *HLA interface* allows the interaction with the HLA coordinator (HLA RTI) for synchronization and communication among the simulation actors during the simulation execution.

ExecutionPlatform: an abstraction representing either a simulator or a physical device, as elements capable of executing system components.

inExecutionComponent: an abstraction representing either a physical device or a model executed in a simulator.

Simulator: a simulator is responsible for executing a subsystem model expressed in a particular modelling language. Several simulation tools may be part of the distributed simulation architecture.

Physical device: a physical device is a physical prototype of a subsystem component. Several physical devices may be part of the distributed simulation architecture. Notice that a physical device is both an *ExecutionPlatform* and an *inExecutionComponent*.

inExecutionModel: a model executed inside a simulator. Notice that differently from physical devices, a model needs to be executed inside a simulator to run and be capable of performing its functionality.

Thing: a "thing" is a service layer to interact with both physical and virtual components (*inExecutionComponents*). "Things" in the IoT jargon are required to be globally addressable, use interoperable protocols and expose services for query and change their state and any information associated with them. We aim at exposing such a representation of the components involved in the simulation of a system. The purpose is to provide a larger interaction of the user with the simulation, which might involve the query of particular data to the components or the modification of component properties.

HLA Coordinator: the HLA coordinator (HLA RTI) is responsible for scheduling the execution of the simulators and physical devices, in order to perform a distributed real-time simulation. It is the responsibility of the coordinator to synchronize the execution of the simulators and the physical devices, so as to keep their internal time aligned. Moreover, the coordinator also provides services that enable the communication among the simulation actors.

SPRINT platform: the SPRINT platform [17] is composed by the SSI and SII layers. The distributed HiL simulation uses the SPRINT platform to setup the simulation.

Simulation Dashboard: the simulation dashboard exposes the services of the distributed HiL simulation platform to the user. Through the dashboard, the user must be able to specify the system model to be simulated, the simulation parameters (e.g., simulation time), control the deployment of the execution responsibilities on the simulation actors, start and control the simulation (e.g., abort the simulation, suspend the simulation), and visualize the simulation results. The simulation dashboard can also be used to access the executed components using the "Thing" abstraction.

The introduction of the representation of executed models and physical devices as "things" in a IoT context creates a new perspective to look at the components involved in a simulation. The IoT view (Figure 6-2) allows accessing physical devices and models in the same manner, as objects living "in the cloud" and abstracting all the mechanisms that are needed for their execution (e.g., the simulators, the HLA coordination mechanisms). "Things" are addressable over the Internet, and expose data over standard protocols. We hence envision the capability of the user of interacting with the IoT view using third-party applications (like http browser) to access the executed components. The SPRINT simulation dashboard will instead enable the user interaction with the services of the simulation platform for setup and management of the simulation. This section will focus on this set of services, so as to define the simulation platform; the description of the IoT view will be addressed in section 12.

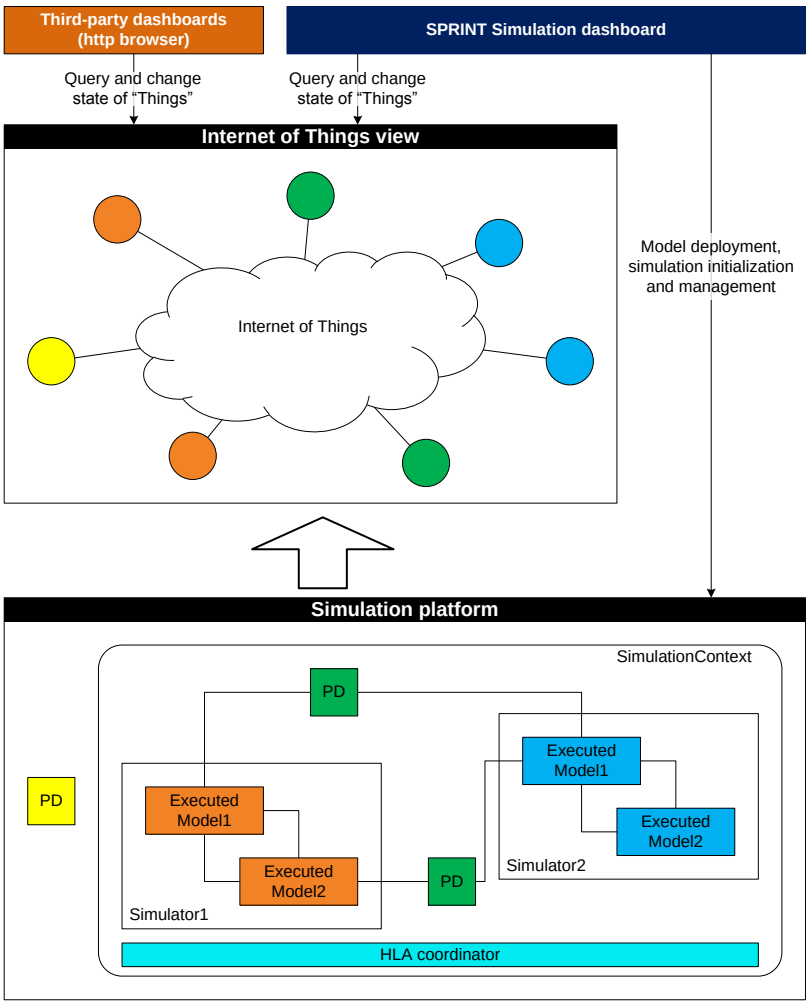

Figure 6-2: Internet of Things view of the simulation platform

6.2 Interaction among the simulation actors

The components in Figure 6-1 are intended to interact through a communication network (e.g., a LAN or the Internet), and which allows them to be executed in a distributed fashion. This subsection describes the interaction that occurs among the components when performing a distributed HiL simulation.

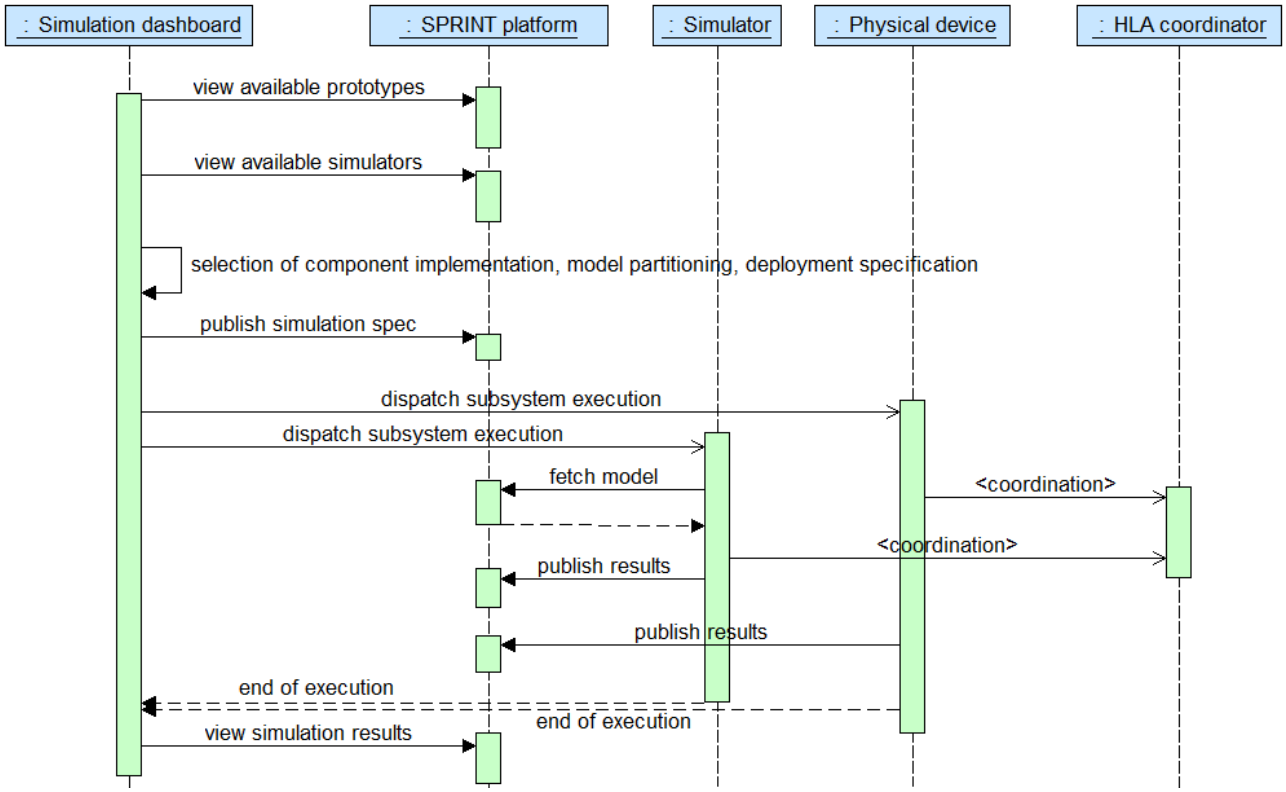

Figure 6-3: Interaction among the components of the Distributed HiL Simulation

A user (Figure 6-3) sits in front of the simulation dashboard to perform the simulation of a particular system. As a first step, the dashboard visualizes the (virtual or physical) prototypes that are available for the system components, shared on the SPRINT platform. Notice that more than one implementation may be available for a particular component. Next, the list of available simulators is retrieved from the SPRINT platform.

With this set of information, the user can select the component implementation to be used in the simulation, how the system model must be partitioned and where the different parts must be executed (deployment specification). This process is a relevant part of the specification of the simulation to be performed. Other parameters (e.g., the simulation time) may need to be specified by the user before starting the simulation. The specification of the simulation to be performed is published into the SPRINT platform.

At this point, the user can start the simulation. The dashboard dispatches the execution of the different components to the simulation actors, as defined by the user. Simulators and physical devices are required to participate to the creation of the simulation federation by interacting with the HLA coordinator. Before the interaction with the HLA coordinator, a simulator may need to fetch the model to be executed if the model is not available in its internal model library.

The execution of the simulation involves several interactions between the simulation actors and the HLA coordinator. At the end of the simulation, each simulation actor publishes the simulation results in the SPRINT platform, establishing a relation between the result set and the simulation specification. This relation is important because it allows tracking under which conditions (e.g., selected models and simulation parameters) a set of results has been generated. Each simulation actor is also responsible for notifying the end of the execution to the simulation dashboard. At the end of the simulation, the user visualizes the simulation results through the simulation dashboard.

6.3 Concrete architecture

In this section, we aim at refining the description of the architecture, focusing on two aspects:

- The deployment of the functional components on computational nodes
- The communication protocols that enable the interaction among the components.

Figure 6-4 shows a legal deployment of the functional components on nodes. We notice that each concrete component in Figure 6-1 can be mapped on a different computational node. The physical device is by itself a node of the network.

A distributed HiL simulation may involve several simulators and physical devices, whereas it requires a single HLA coordinator; a HLA coordinator may eventually serve more than one HLA federation. The SPRINT platform is deployed in the picture on a single node; however, it can be distributed on several nodes, as shown in [16]. The simulation dashboard is the tool that allows a user to set up the simulation and start it. The simulation specification is shared on the SPRINT platform, hence its definition may derive from the collaboration among different people. In this scenario, multiple instances of the simulation dashboard, deployed in different locations, may participate in a distributed HiL simulation. The start-up of the simulation must be managed by a unique simulation dashboard. An installation of the simulation platform may hence involve several nodes. It is obviously possible to host more than one functional component on a single node.

The nodes are interconnected through the Internet. The communication among the nodes can be partitioned in three logical data flow:

1. Communication with the SPRINT platform, which involves Simulators, physical devices and the simulation dashboard. The SPRINT platform exposes its functionalities through RESTful web services; the client nodes communicate with it through HTTP over a TCP/IP link.

2. Communication between the dashboard and the simulation actors (simulators and physical devices). Simulator and physical devices expose a *service interface* (Figure 6-1) through web services; the dashboard hence controls their execution through a HTTP connection over a TCP/IP link.

3. Communication between the simulation actors and the HLA coordinator (HLA RTI). At the application level, HLA defines its own protocol for communication, whereas it supports the usage of different protocols at the transport level. The user may select a reliable or best effort communication, which is mapped to an available transport service by the HLA layer. Some implementations of HLA (e.g., CERTI RTI [18]) support the use of TCP and UDP at the transport level, respectively used for reliable and best-effort communications. The HLA coordinator listens for connections from the simulation federates on a user configurable pair of TCP and UDP ports.

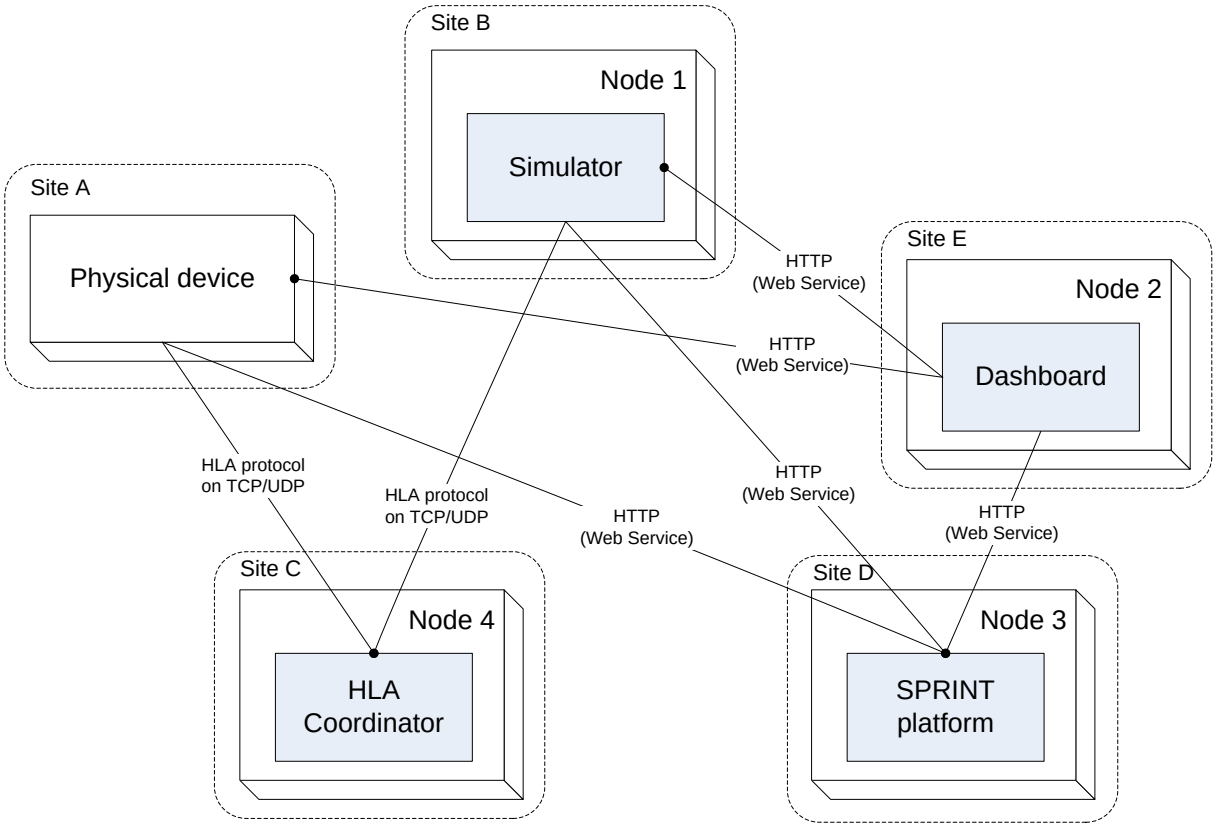

Figure 6-4: Deployment of functional components on nodes and communication protocols

In Figure 6-4 we have supposed that the different nodes are in different sites. In the SPRINT scenario, each site is likely to be a company involved in the design of a particular system. Companies apply rules to control the incoming and outcoming network data flow in their network, and it is hence important to analyse the network security exceptions that a particular deployment of the distributed simulation platform components

may generate. For this purpose, we have summarized in Table 6-1 the role that each component plays in the communication, client or server, and the protocol used for communication.

In general, firewalls are configured to not prevent outgoing http connections on port 80 and 443 for access to web-pages. Web-services use http and can hence pass through firewalls without explicit configuration [19]. All the communications in Figure 6-4 based on HTTP should not demand a dedicated configuration at the client site. The server must instead be located at a site where incoming HTTP connections can pass through.

The HLA communication may require security exceptions on both the client and server side. Firewalls prevent the exchange of data based on the HLA protocol (application firewalls) on arbitrary TCP and UDP port; application level firewalls can also check the protocol used for the communication, and block outgoing connections on open ports (80 and 443) when the application protocol is not the expected one.

Table 6-1: Components, protocol and communication role

	Client	**Server**
Simulator	HTTP, HLA	HTTP
Physical device	HTTP, HLA	HTTP
HLA coordinator	-	HLA
SPRINT platform	-	HLA
Dashboard	HTTP	-

HLA implementations exist (e.g., CERTI RTI [18]) which support the use of HTTP tunnelling for communication between the simulation actor and the HLA coordinator. In this case (Figure 6-5), the simulation actor (physical device or simulator) may cross the site boundaries encapsulating the HLA data in a HTTP communication. The HTTP flow is received by a HTTP proxy that forwards the communication to the HLA coordinator using the HLA protocol. The HTTP proxy may be hosted at a different site where less stringent security rules are required.

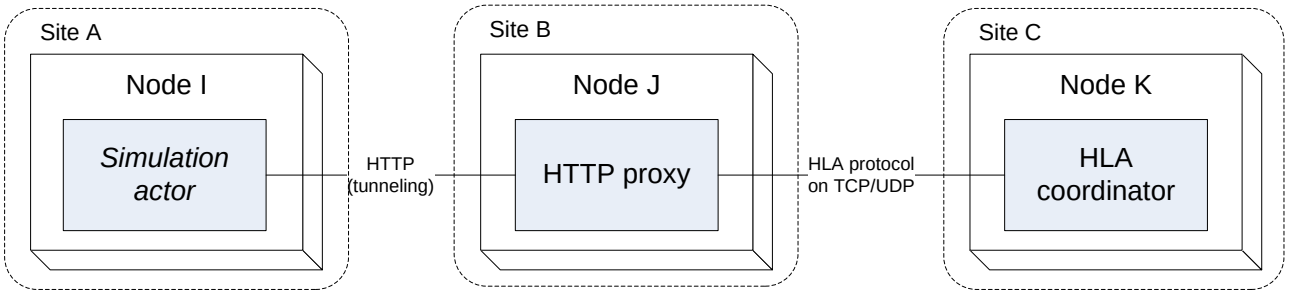

Figure 6-5: HTTP tunnelling of HLA communication

Both simulators and physical devices need extra functional components (interfacing components) to be integrated with the simulation platform, as shown in Figure 6-1: the HLA interface, the Service Interface and the SPRINT interface. Figure 6-6 shows a typical deployment of these functional units. In the case of simulators (Figure 6-6-a), all the functional components can be hosted on the same node, eventually as part of the same application process. This integration minimizes the overhead of the communication between the simulator and the interfacing components.

Physical devices may not have in general computational capabilities to host the execution of the interfacing components. We hence envision the introduction of an additional node (Figure 6-6-b) executing the interfacing components, this node is identified as the SPRINT gateway. The protocol that rules the communication between the two nodes depends on the available interfaces of the physical device. Figure 6-7 shows, as an example, that data required for the registration of the physical device on the SPRINT platform may arrive through a RFID reader, which scans the tags applied on the available devices.

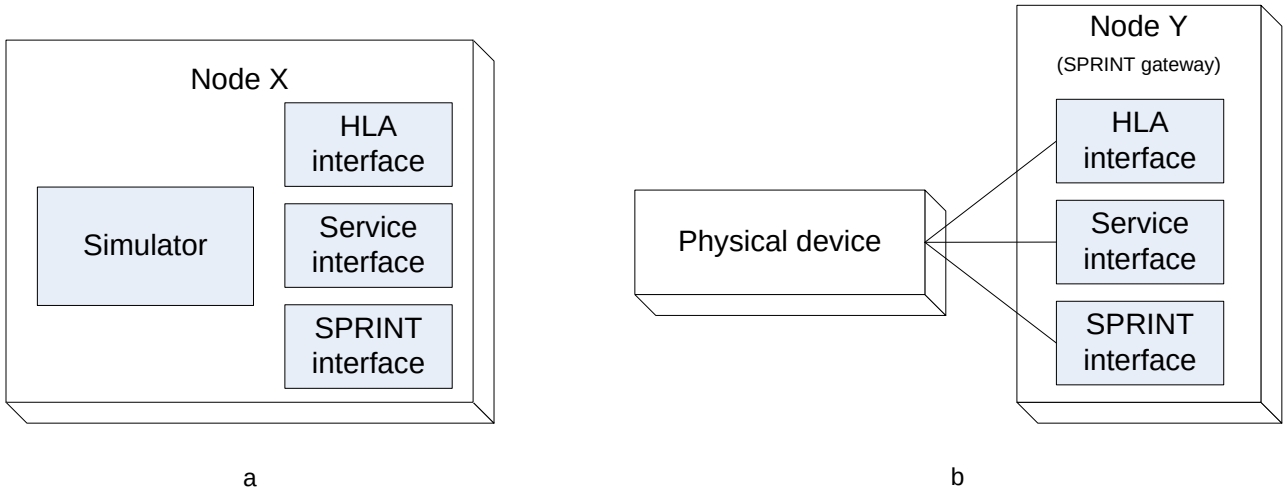

Figure 6-6: Deployment of functional units of a simulation actor. Simulator (a) and Physical device (b).

Figure 6-7: Registration of physical device using RFIDs

7 SPRINT platform simulation services

The SPRINT platform is involved in different phases of the distributed HiL simulation:

1. Selection of component prototypes to be used in the simulation. The SPRINT platform contains the list of all the available component prototypes (models and physical devices) that have been previously registered, as well as the list of partitioning points (see section 7.2) where the system model can be “broken” for distribution on different simulation actors.
2. Selection of the simulation tools to be used. Simulation tools are registered in the SPRINT platform to enable the discovery of the provided services.
3. Publication of the simulation specification using the simulation dashboard. The simulation specification describes a particular simulation run (e.g., component implementations, parameter values).
4. Fetching of the model to be executed by a simulator. A simulator may fetch the model to execute from the SPRINT platform.
5. Publication of simulation results. A simulation actor publishes the simulation results generated locally into the SPRINT platform. The simulation results are linked to the related simulation specification.
6. Inspection of the simulation results. The user accesses the simulation results stored in the SPRINT platform using the simulation dashboard.

The exchange of models (point 1 and 5) is already in the set of the use cases considered so far for the specification of the SPRINT platform. OSLC-AM [20] is the candidate technology for sharing models among tools, supported by the semantic mediation services. Model sharing has focused so far on the component structure (component interface, interconnection among components, and hierarchy), and has not yet considered the exchange of simulative (executable) model implementations. Hence, there are some operations that are not currently covered by the SPRINT platform specification, and that pose additional requirements on the kind of information that the platform allows sharing:

- Executable models of components (virtual implementations of components) (point 1 and 4)
- Partitioning points (point 1)
- Available physical devices (physical implementations of components) (point 1)
- Available simulators (point 2)
- Simulation specification, to identify a particular simulation run (point 3)
- Simulation results, linked to a particular simulation specification (point 5)

Services to publish and retrieve these resources must be provided by the platform. The following sections analyse the introduction of support for these features in the SPRINT platform.

7.1 Sharing of executable models

In the SPRINT platform the behaviour of models is exchanged as virtual implementations. A virtual implementation is the executable version of a model, for example in the case of Modelica it could be an executable or FMU generated from the Modelica code. Model implementations must be in relation with the structural description of the model. Multiple model implementations can be in relation with a single component, as multiple views of the component behaviour.

The meta-model for virtual implementations is shown in Figure 7-1. A virtual implementation can be in a standardized format like an FMU or in a tool specific format. If it is in a tool specific format the original tool is most likely required to be able to run the simulation, whereas if it is in a standardized format a general purpose simulation tool might be used to run the simulation. The implementationType attribute specifies the format of the model. The list of supported implementation types in SPRINT is provided in the homonymous enumeration. Since a virtual implementation in many cases will consist of code compiled for a specific platform it has restrictions on where it can run. This restriction is covered by the executionPlatform attribute. Specific implementation types may need to be a specialisation of a virtual implemention with its own attributes. Initially the only virtual implementations that will be handled by the distributed HiL platform are FMUs, a specialization for FMUs are shown in the figure. It contains the necessary attributes to match the FMU with a simulator that is capable of simulating that particular FMU, i.e. what version of FMI and if it is a Model Exchange FMU or a Co-Simulation FMU.

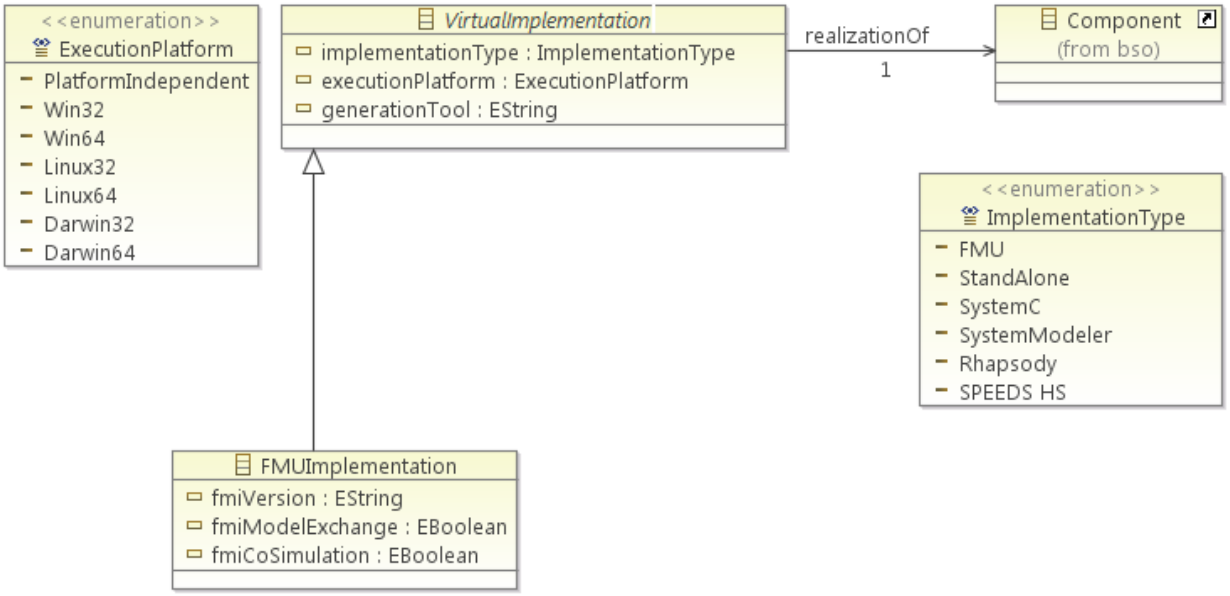

Figure 7-1: Meta-model for virtual implementations.

7.2 Support for model partitioning

For execution on a distributed simulation platform, a system model must be partitioned in subsystem units and deployed for execution on different federates. Components deployed on different federates need to communicate through HLA, in terms of sending/reception of interactions and object attribute updates.

We can identify two scenarios for model partitioning:

1. The partitioning is performed on the interface between two components (Figure 7-2 a). An HLA proxy maps the communication on the component interface into HLA communication mechanisms.

2. The partitioning is performed by splitting a component into subcomponents (Figure 7-2 b), each capturing part of the original component functionality plus the HLA communication mechanisms.

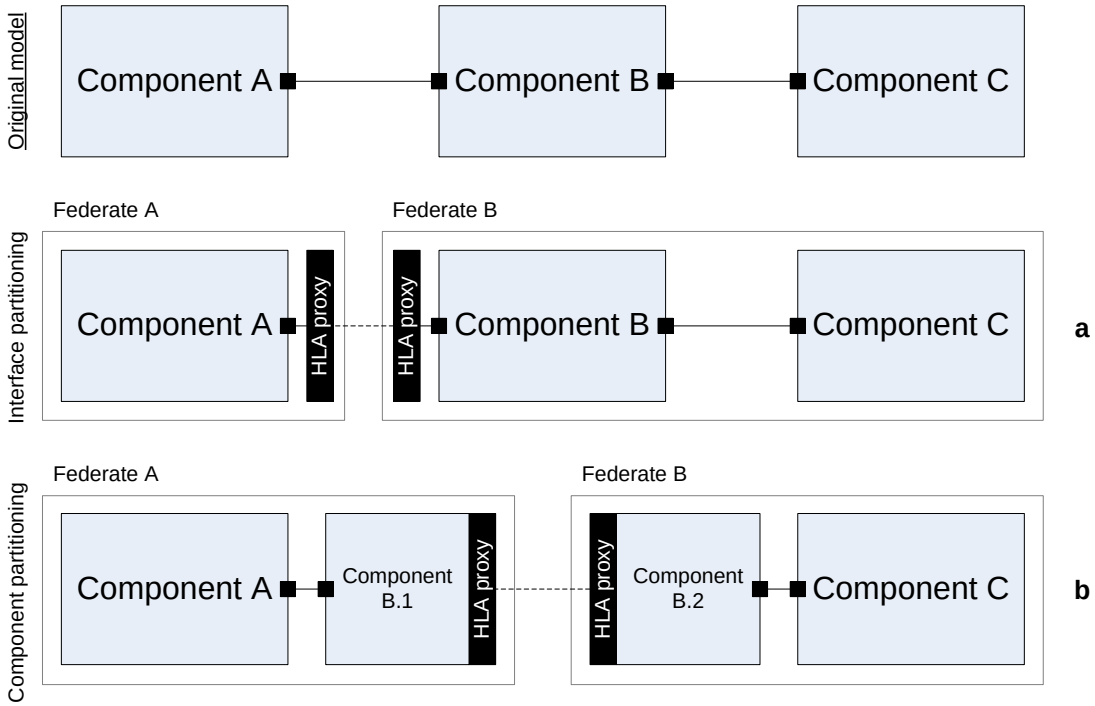

Figure 7-2: Partitioning of a model

Partitioning a component requires to modify both the architecture and the implementation of the model: a single component is replaced by a set of components, and their implementation must be HLA aware. Partitioning a model on the interface is hence more convenient. We will focus hereafter on this scenario. We can identify three alternative approaches to implement the HLA proxy (Figure 7-3):

a) The proxy is external to the component. No modification of the original component is needed, both in term of interface and implementation. The HLA proxy can be defined out of the interface description of the component; therefore, no knowledge of the component internals is needed for partitioning.

b) The proxy is embedded in the component. The usage of an embedded proxy requires modifications of the component interface and implementation. On the interface, ports for communication with the remote component are no more needed; communication occurs by invoking HLA services. On the implementation, data to be communicated must be mapped on HLA object classes and interactions. Nevertheless, the (partial or complete) awareness of the internal behaviour of the component provides more opportunities for optimizing the implementation of the proxy; the HLA interface can be designed so as to reduce the data exchange with the remote peer, and/or to maximize the lookahead value [21].

c) The proxy is external to the component, but interacts with the simulation kernel. This approach aims at keeping the component implementation unmodified and independent from the HLA platform, while acquiring a partial awareness of the component behaviour using the service primitives provided by the simulation kernel. For example, in a discrete event simulation environment the HLA proxy might retrieve the list of events scheduled by the component from the event queue in the simulation kernel. This approach is the candidate for adoption in the SPRINT simulation platform; its potentials and limitations will arise from its application to the SPRINT simulation use cases.

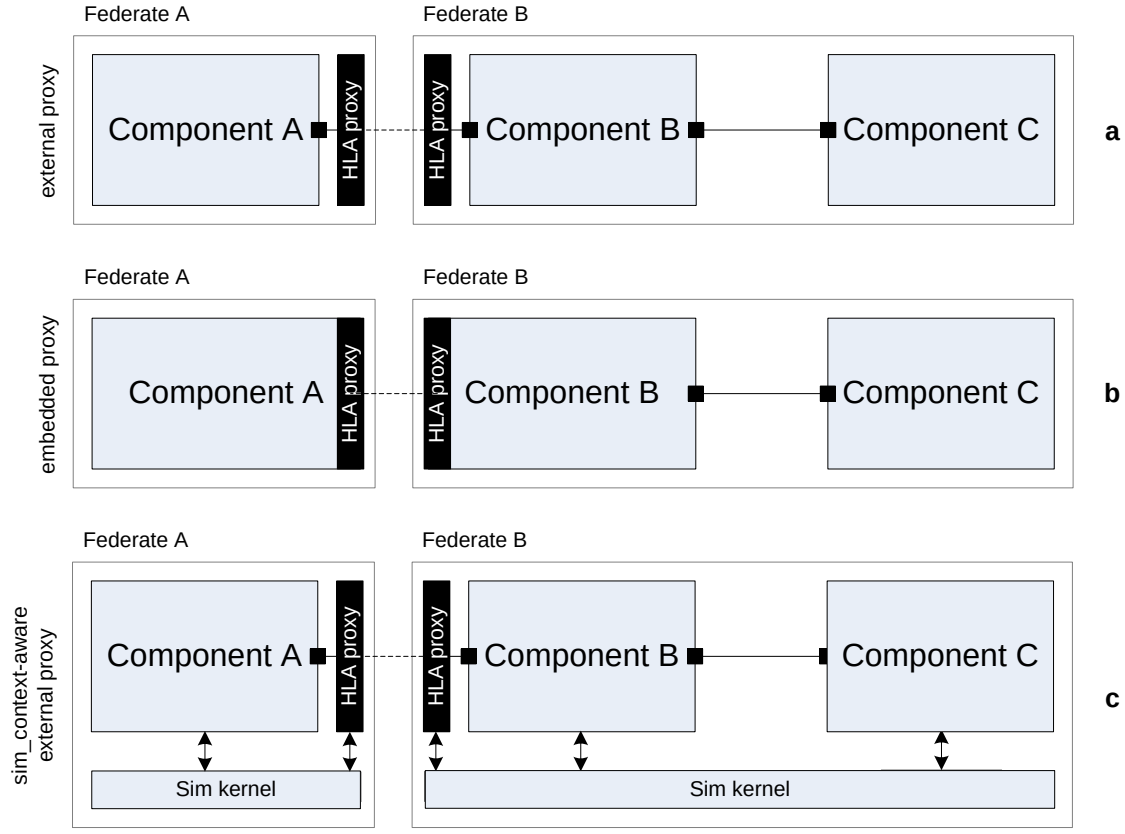

Figure 7-3: Alternatives for the HLA proxy implementation

All the approaches described so far require an adaptation of the model for its partitioning, which is subject to the availability of proper HLA proxies (for the interface to be partitioned) or of HLA-aware components. It should be possible to automate the generation of HLA proxy, so as to deploy the subsystem units for execution without human intervention. In this deliverable, we will not investigate this approach. We will assume that these extra model elements are defined as part of the modelling process and must be shared as well through the SPRINT platform; their availability must guide the user in the partitioning process. In the modelling phase, a user will define and share HLA proxies or HLA-aware components. Then, in the model partitioning, the user will decompose the system breaking down an interface for which a “HLA partitioned implementation” is available.

7.3 Registration of physical devices

SPRINT aims at supporting both models and physical devices as (prototype) implementations of a system component. The SPRINT platform must hence provide the capability to register the availability of a physical device, and to link the representation of the physical device to a component in the system model.

Unlike a simulation model, a physical device is by itself a simulation actor; it can interact with the SPRINT platform and must expose services to start its execution and involve the device in a distributed simulation. Moreover, physical devices that can be stalled (see section 9 for details) do not require a wall-clock time execution of the simulators.

The SPRINT platform must allow the registration of the following information for each physical device:

- URI to access the physical device services
- Stallable [true|false]. Based on the value of this parameter, the simulation platform decides whether to adopt a wall-clock time execution.

The complete process for register physical devices is described in [29].

7.4 Simulation specification

The simulation specification describes the particular simulation to be performed on the distributed HiL simulation platform. The user defines the simulation specification using the simulation dashboard. Figure 7-4 shows the information that a simulation specification must contain.

A distributed simulation specification is given by a set of deployments. A deployment specifies the implementation that must be used in the distributed HiL simulation for a particular component instance, represented in the diagram by a componentProperty (element of the BSO ontology [22]), and where this implementation is executed. We identify two specializations of the deployment class:

- A VIDeployment, which associates to a componentProperty a VirtualImplementation, namely an executable model of the component. The deployment specifies as well the particular simulator where the model is intended to be executed.
- A PDDeployment, which associates to a componentProperty a PhysicalDevice.

Both deployments are associated with a set of configuration information, providing values for the parameters of the physical device (PDDeployment) or of the model (VIDeployment). Parameters are provided in the form of parameter name and value pairs.

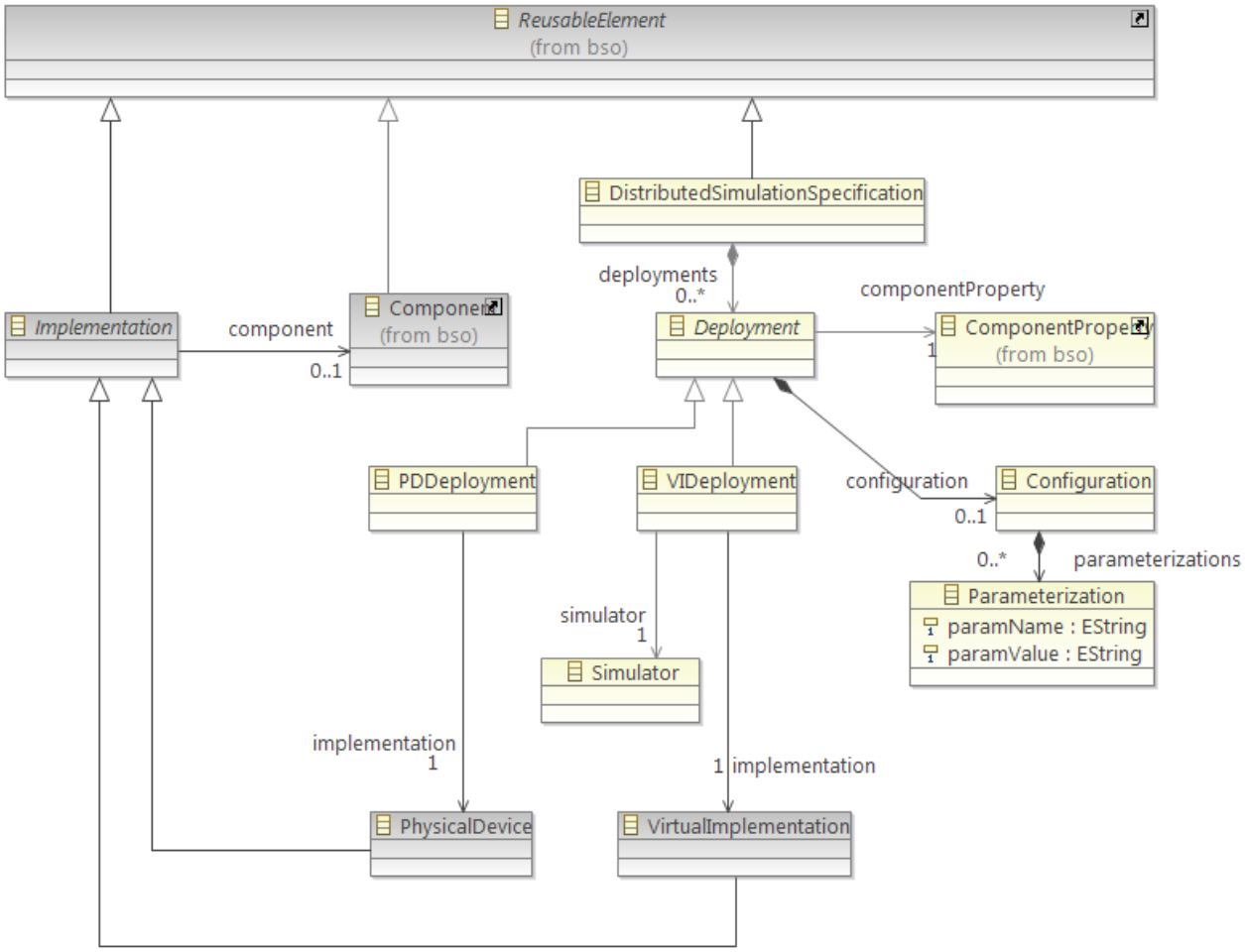

Figure 7-4: Meta-model for Simulation specification

7.5 Simulation tools

Simulator tools need to register on the SPRINT platform to be discovered during the simulation specification phase. Users will define the deployment of the models based on the available simulators and their capabilities.

Figure 7-5 shows the information published to describe an available simulator:

- A Host, which defines where the simulator is running. More in detail, three attributes are provided:
  - URL, which is used to access the simulator services. As pointed out in Section 6.3, a simulator exposes a set of web services that can be remotely invoked. The URL provides the point of access to these services.

- Location, which is a string providing a human-readable description of where the simulator is running.
- Performance, which describes the computational capabilities of the hosting machine of the simulator.

- A set of ImplementationTypes, which defines the types of models that the simulator is capable of executing. These attributes are intended to guide the user in the model deployment phase, so as to assign a model to a simulator supporting the execution of the particular model format.

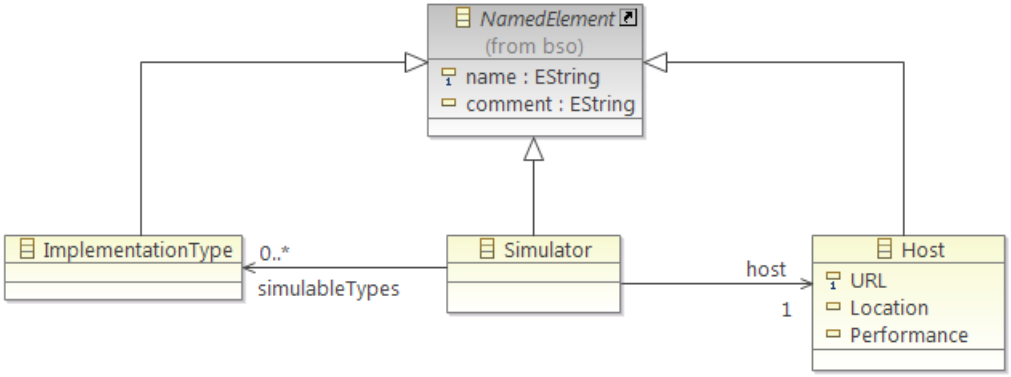

Figure 7-5: Meta-model for Simulators

8 Simulators

This section provides the specification for integrating a simulator into the SPRINT platform, and describes the simulators that will support the distributed HiL simulation during the project. For each simulator, the envisioned architecture of the integration will be provided, to serve as application examples of the specification.

8.1 Specification for platform integration of a simulator

A simulator involved in a distributed HiL simulation must interact with the other simulation agents (Figure 6-3). The set of interactions can be divided in two groups:

- Interactions for setting up the simulation (simulation setup)
- Interactions for run-time coordination with the other simulators and physical devices (simulation execution)

The following subsections will provide the specification of the simulation interface to serve both sets of interactions.

8.1.1 Specification for simulation setup integration

A simulator in the SPRINT distributed HiL simulation architecture is a resource accessible over the Internet. The user interacts with a simulator using the simulation dashboard. Once the user requests to execute a model, the simulator needs to fetch the model from the SPRINT platform, run it and then publish the results into the SPRINT platform. At this point the end of the simulation is notified to the user. The simulation setup flow described in Section 6.2 shows that there are two kinds of interactions involved in the setup of a simulation:

- The interaction between the user, through the simulation dashboard, and the simulator (service interface)
- The interaction between the simulator and the SPRINT platform (SPRINT interface)

Both a simulator and the simulation dashboard provide a set of services, and they interact invoking each other's services. A list of the services provided by the simulation and the dashboard follows.

Services provided by the simulator

- void initSimulation(URI userID, URL simulationNetlist, URI simulationID, bool federationCreator)
  - *Description:* a service to involve a simulator in a distributed simulation. See section 8.1.2 for additional details.
  - *Parameters:*
    - URL userID: the identifier and locator of the user requesting the service. It must be the URL of the simulation dashboard.
    - URL simulationNetlist: the identifier and location of the simulation netlist to be executed in the simulator, used to fetch the required resources (netlist and component model implementations) from the SPRINT platform.
    - URI simulationID: the identifier of the distributed simulation (HLA federation). All the simulators involved in the same distributed simulation must have the same simulationID.
    - bool federationCreator: set to true when the simulator is responsible for starting the federation.
- void startSimulation(URI simulationID, double simTime)
  - *Description:* a service to start a simulation just initialized or in suspended state. After starting a simulation, simulated time starts to progress. The service must be called on the simulator acting as federation creator.
  - *Parameters:*
    - URI simulationID: the identifier of the distributed simulation (HLA federation)
    - double simTime: the simulated time the simulation must reach before suspending (in seconds). The value -1 means that the simulation will suspend when an explicit stop will be issued.
- void stopSimulation(URI simulationID)

- *Description:* a service to suspend a simulation during its execution. A suspended simulation can be resumed using the startSimulation service.
- *Parameters:*
  - URI simulationID: the identifier of the distributed simulation (HLA federation)

- void endSimulation(URI simulationID)
  - *Description:* a service to end a simulation while in suspended state. After the endSimulation is called, the simulation cannot be resumed. Each simulation actor shares the simulation results.
  - *Parameters:*
    - URI simulationID: the identifier of the distributed simulation (HLA federation)
- void abort(URI simulationID)
  - *Description:* a service to abort a simulation in progress. The service must be called on the simulator acting as federation creator, and must cause the whole simulation to stop.
  - *Parameters:*
    - URI simulationID: the identifier of the simulation to be aborted

Services provided by the dashboard

- void readyToStart(URI simulatorID, URI simulationID)
  - *Description:* a service required by each federate to notify the completion of the simulation setup process.
  - *Parameters:*
    - URI simulatorID: the identifier of the simulator issuing the callback
    - URI simulationID: the identifier of the federation that the simulator has joined
- void simulationStopped(URI simulatorID, URI simulationID)
  - *Description:* a service required by each federate to notify that the simulation has got suspended
  - *Parameters:*
    - URI simulatorID: the identifier of the simulator issuing the callback
    - URI simulationID: the identifier of the federation that the simulator is involved in.
- void simulationEnded(URI simulatorID, URI simulationID, URI simulationResults)
  - *Description:* a service required by each federate to notify the end of the simulation and the URL of the simulation results
  - *Parameters:*
    - URI simulatorID: the identifier of the simulator issuing the callback
    - URI simulationID: the identifier of the federation that the simulator was involved in
    - URL simulationResults: the identifier and locator of the simulation results, to be used by the user to access them
- void error(int errNo)
  - *Description:* a service required by a federate to notify an error in performing a required service
  - *Parameters:*
    - int errNo: the error code that identifies the error just occurred. A set of standard error codes will be defined in the next version of the deliverable.

The definition of a new OSLC specification for simulation will be considered to standardize the set of services for interaction with a simulation actor. The implementation of the service interface will be based on web services.

Simulators need to interact with the SPRINT platform to register their availability, fetch the simulation models and publish the simulation results. The data format and interchange technology is indicated by the underlying technological platform and are discussed in the related deliverables [17]. Currently, the SPRINT platform supports the OSLC [23] technology (data format and communication protocol) for the data interchange and therefore simulators shall comply with it.

8.1.2 Specification for run-time integration (HLA)

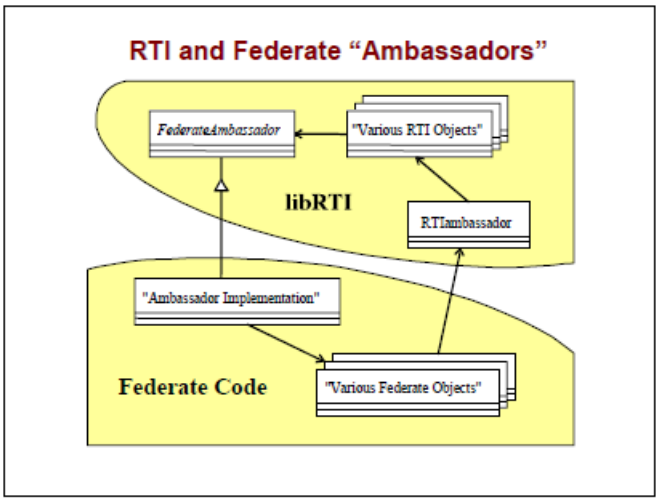

Figure 8-1: General architecture for integration of a federate in HLA

A simulator can be integrated with the HLA platform by the means of a libRTI [24]. The libRTI provides the mechanism for communication between the simulator (playing the role of a federate) and the HLA coordinator (RTI), in the form of two classes:

- RTIambassador, which exposes the services of the RTI to be invoked by the federate
- FederateAmbassador, which implements the callbacks invoked by the RTI to communicate with the federate. The actions to be performed when a callback is issued may depend on the particular simulator being considered. The simulator is hence expected to define its own implementation of the FederateAmbassador, using the implementation in the libRTI as base class.

The execution of a simulation over HLA involves three main activities:

- The creation/management of the federation
- The time synchronization among the federates
- The communication among the federates

This section aims at specifying the approach to be used to perform these activities using the services of the HLA platform.

Creation/management of the federation

Figure 8-2 shows the interaction between the simulation dashboard and the simulation actors, and the corresponding interactions that occur between the simulation actors and the HLA coordinator. All the simulation actors involved in a distributed simulation using HLA become federates of a federation.
All the federates in the simulation must perform the same actions at start-up. The dashboard invokes the initSimulation service on all the simulators that will become part of the federation. Once this service gets invoked, the federate tries to create the federation. In case the federation exists because another federate has already created it, the federate gets a specific exception and moves to the join process. After joining the federation, the federate sets its timing behaviour (regulating and/or constrained, together with its lookahead) and tries to registers a synchronization point with tag “Init”. The scenario in which the synchronization point has been already registered is managed as for the federation creation.

The synchronization point allows aligning all the federates that join the federation before starting to advance the simulation time. Once the synchronization point is successfully set and is “announced” to the federate, the federate invokes the readyToStart service of the simulation dashboard. The dashboard waits for receiving a ready to start by all the federates, before invoking the startSimulation service. The federate receiving this service invocation communicates to the HLA coordinator that the synchronization point has been achieved. As defined by the HLA standard, when all the federates achieve the synchronization point, a federationSynchronized callback is issued on all the federates, that can hence start the simulation.

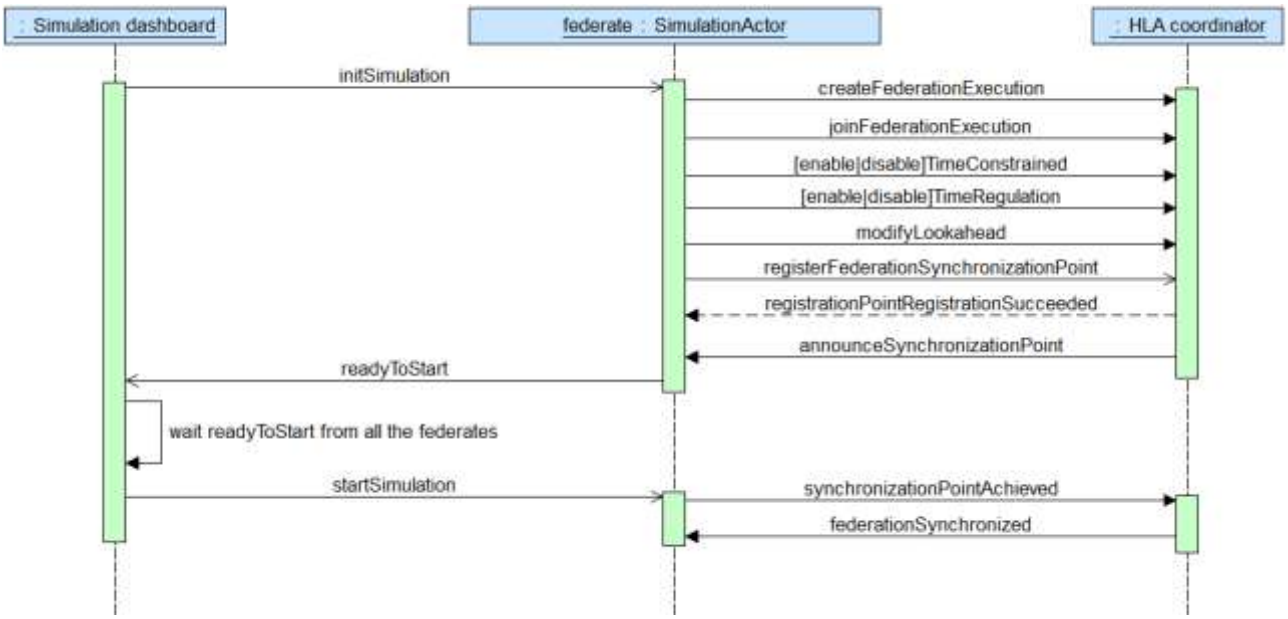

Figure 8-2: Required interaction for creation of a distributed simulation using HLA

Time synchronization among the federates

HLA provides mechanisms for both optimistic and conservative time synchronization (section 2.1.1). The mechanism adopted by a particular federate is completely transparent to the other federates. There is hence no need to dictate a particular time synchronization approach for use in the SPRINT distributed HiL simulation. All the time synchronization approaches are admissible.

Federates may be time regulating and/or time constrained. The role played by a simulator depends on the model being executed by the federate, as well as by the interaction of the model with the other subsystem. A set of guidelines are provided for the admissible scenarios:

- Neither time regulating, nor time constrained: applicable to simulators executing untimed models, or simulator executing models that have no interaction with other subsystem prototypes.
- Time regulating: applicable to simulators executing timed models with uni-directional interaction with the other subsystem prototypes, where the model acts as source of the interaction.
- Time constrained: applicable to simulators executing time models with uni-directional interaction with the other subsystem prototypes, where the model acts as destination of the interaction.
- Time regulating, time constrained: applicable to simulators executing timed models with bi-directional interaction with the other subsystem prototypes.

The execution of models together with physical devices may require mechanisms to synchronize the speed of execution with the wall-clock time. In the scenario where the simulator is time constrained and the physical device is time regulating, the synchronization is guaranteed by HLA: the simulator will not run faster than the physical device. The opposite scenario, simulator as time regulating and physical device as time constrained, is instead more challenging: the physical device cannot (in general) constrain its execution. In this scenario, it is still up to the simulator to progress time at the wall-clock time pace, so as to stay synchronized with the devices in the real world. There may be scenarios in which simulators cannot execute the models fast enough to guarantee the synchronization (e.g., because the models are computationally complex), and HiL simulation is hence hardly applicable.

Communication among the federates

The communication among federates in a distributed HiL simulation enables the interaction among the component prototypes executed in a distributed manner. Components that are interconnected in the system model and are deployed for execution on different simulation actors need to exchange the data flowing on the interface using the HLA communication mechanisms. Approaches to partition a model and adapt the interface to the HLA primitives have been already described in section 7.2. All the approaches involve the definition of components that expose an HLA interface for communication with the remote peers. Simulators are hence required to execute these components to guarantee the communication of the locally executed model with the remote ones.

Notice that there is a strong relation between the required interaction among the models and the synchronization approach to be used. In particular, the use of Time Stamp Order (TSO) messages may require a federate to adopt a time regulating and/or time constrained role. Moreover, the lookahead value constrains the timestamp of sent messages to be in the future with respect to the current time; generation of messages with timestamp set at the current time requires the usage of a particular set of synchronization primitives.

9 Physical devices

This section provides the specification for integrating a physical device into the SPRINT platform, and describes the classes of devices that will be considered for integration. Several analogies exist with the integration of simulators described in section 8. This section will point out the relevant differences.

9.1 Specification for platform integration

As for simulators, a physical device that participates in a distributed HiL simulation must interact with the other simulation actors to support the simulation setup and the simulation execution. From an architectural standpoint, we envision the definition of a SPRINT gateway (Figure 9-1) that manages, on the Internet side, the interaction with the other simulation actors, and controls accordingly the physical device. Notice that the interface between the gateway and the physical device depends on the particular device being considered; it may be an electrical connection for simple devices like sensors or actuators, or a network communication (e.g., over CAN bus) for networked devices.

All the interactions described in the next subsections are hence performed by the SPRINT gateway, which is responsible for translating them in a format compatible with the input/output interfaces of the device.

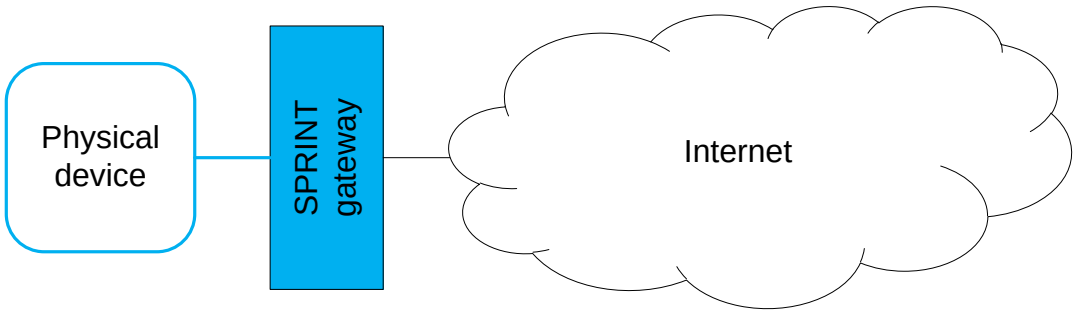

Figure 9-1: Architecture for integration of a physical device in the SPRINT distributed HiL simulation

9.1.1 Specification for simulation setup integration

A physical device in the SPRINT distributed HiL simulation architecture is a resource accessible over the Internet, which provides a set of services. As for models, physical devices must be “published” in the SPRINT platform as implementations of a particular component to enable their discovery by the user. Information published in the platform must contain a URI of the device where the provided services are accessible. The user accesses these services using the simulation dashboard to involve a particular device in the execution of a distributed HiL simulation. Once the simulation completes, the physical device is responsible for publishing any result traces that it might have generated during the execution.

As already seen in section 8.1, we can identify two sets of interaction to support the setup of a distributed HiL simulation:

- The interaction between the user, through the simulation dashboard, and the physical device (service interface)
- The interaction between the physical device and the SPRINT platform (SPRINT interface)

Both interactions are performed in a similar manner to simulators.

The interaction with the user relies on the same set of services described in section 8.1.1.

Physical devices need to interact with the SPRINT platform to register their availability as component implementation and publish the execution traces generated by the device during its execution. The data format and interchange technology is indicated by the underlying technological platform and will be discussed in the related deliverables [17]. Currently, the SPRINT platform supports the OSLC [23] technology (data format and communication protocol) for the data interchange and therefore physical devices shall comply with it.

9.2 Real-time synchronization

In a HiL simulation scenario, physical devices and simulators execute together and exchange data. Physical devices execute in real(-world) time (wall-clock time), whereas simulators have an artificial notion of time (virtual time), which is under control of the simulator (time progression can be suspended and resumed). A

HiL simulation platform must aim at synchronizing the physical device time and the simulator time, so as to guarantee that data exchanged between the two domains are injected in the other side at the right point in time. As an example, a command sent by a simulated controller at its virtual time t to a physical actuator should reach the device when its real time is at time t. A measurement of a physical sensor taken at time t' in the real time must be injected in the simulation domain at its virtual time t'. This is the ideal behaviour.

In reality, synchronization between the physical devices and the simulators may experience some errors. The major cause is the possible slow-down of the simulation execution. Simulators might not be capable of executing models at the wall-clock time pace, due to the computation complexity of the models. As an effect, the virtual time may lag behind the real time. Moreover, in the SPRINT scenario where simulators and physical devices communicate over the internet, communication delays prevent a precise synchronization between remote simulators and physical devices.

9.2.1 Synchronization approach

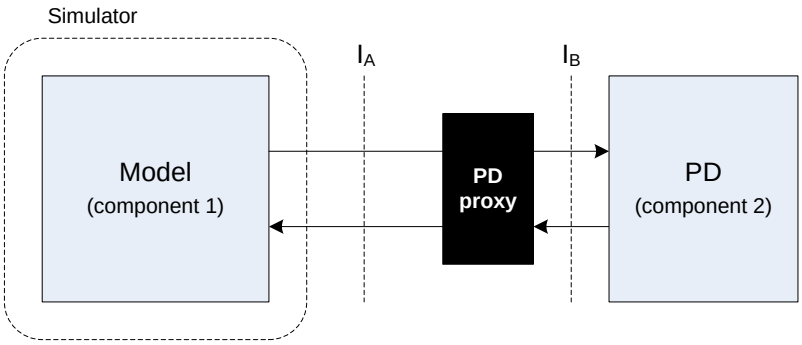

Figure 9-2: Case study for the real-time synchronization

The synchronization approach that will be adopted in the SPRINT simulation platform will be explained in this section using a simple case study with a simulator and a physical device, shown in Figure 9-2. A PD proxy interfaces the device with the virtual (simulated) part of the system. We assume that the model and the PD proxy share the same notion of virtual time, whereas the physical device runs in real-time. Both the virtual time (t_v) and the physical device time (t_r) are initialized at zero at the beginning of the simulation. The PD proxy can evaluate the value of t_r at any t_v accessing the physical device clock to measure how much real time has elapsed since the beginning of the simulation; alternatively, the proxy can estimate t_r using the clock of the node on which it is executed. We will indicate with $\Delta t_s(t_r)=t_r-t_v$ the difference between the physical device time and the virtual time at a certain (real) time t_r; notice that $\Delta t_s(t_r)$ is always larger than or equal to zero.

We consider two scenarios:

- The model sends a message to the physical device at $t_v=t_1$. The message reaches the PD proxy at time $t_v=t_1$; the proxy should in turn deliver it to the physical device when $t_r=t_1$. As already pointed out, t_r is larger or equal than t_v. The proxy can hence deliver the message no earlier than $t_r=t_v+\Delta t_s(t_r)$.
- The physical device sends a message to the model at time $t_r=t_2$. The message reaches the proxy when t_v is less than or equal to t_2. The proxy can hence deliver the message to the model at the right point in time, namely when $t_v=t_2$.

We notice that an error is introduced in the simulation when data are transferred from the virtual domain to the physical domain, in the form of an extra delay. The HiL simulation of this simple system with two components would generate a trace equivalent to that of the system in Figure 9-3, where a delay module is introduced on the link between component 1 and component 2. The delay is equal to the difference between t_v and t_r, and it is hence time-varying. It is responsibility of the simulation platform to try to minimize the amount of the delay; however, a perfect synchronization is in general not attainable.

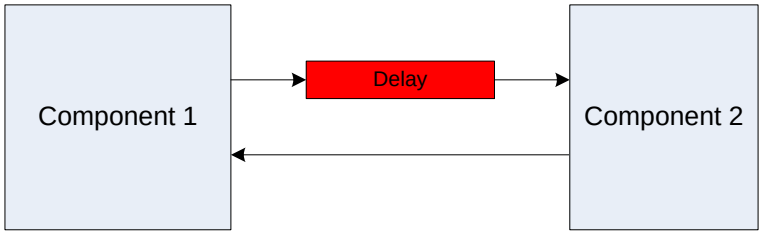

Figure 9-3: Actual simulated system

The user is left with the option of taking into account this delay to guarantee that the timing of the events injected in the physical devices is consistent with the rest of the model. Let consider a scenario with 3 components (Figure 9-4). The output of component 1 is now injected in component 2 and component 3. If we simulate the system executing together models of component 1 and 3 and a physical prototype of component 2, it happens that the data sent by component 1 are injected in component 2 and component 3 at different times, as an effect of the delay in the interaction with the physical device. If the amount of the delay was known, the user might intentionally introduce a delay module on the link towards component 3, so as to align the timing of the inputs to component 3 and component 2. Figure 9-5 shows the simulated model with the intentional introduction of a delay module.

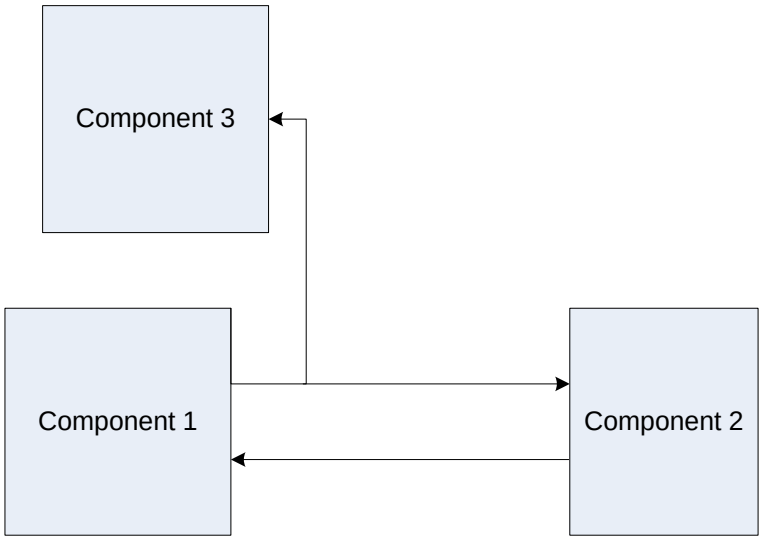

Figure 9-4: Case study with three components

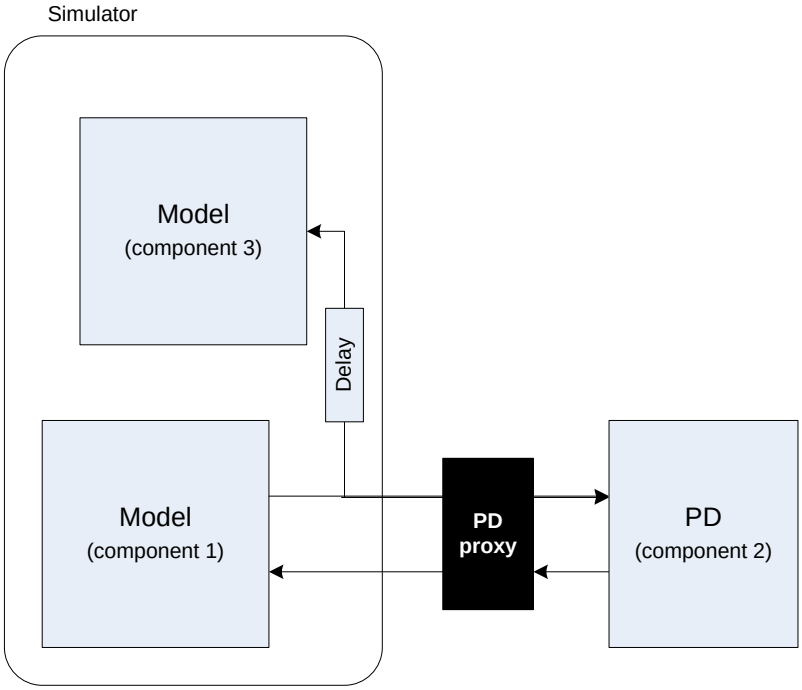

Figure 9-5: Simulation with modelling of HiL delay

The difference between t_r and t_v, which determines the amount of the delay, is a-priori unknown and may vary over time. The PD proxy can measure it, and could then communicate it to the other simulators. However, while the delay value is propagated to the other simulators, the time of the physical device progresses and makes the measurement not accurate; moreover, communicating the delay back to the simulators would introduce an overhead in the simulation execution. We suggest adopting a constant delay value T, configurable by the user. T must be an upper bound of the synchronization error $\Delta t_s(tr)$, i.e., $\Delta t_s(tr) < T$ at any time during the simulation. When a message scheduled at time $t_v=t_1$ arrives to the PD proxy, it will not immediately deliver it to the physical device, but it will wait up to the point in which $t_r=t_1+T$. Notice that if the physical device is already in the future with respect to t_1+T, $\Delta t_s(tr)$ has exceeded the user defined upper bound T. The PD proxy may take an action when this event occurs, like logging the violation or aborting the simulation. In alternative to using a constant upper bound value, the user may specify it as a function of the virtual time ($T=f(t_v)$); this allows increasing the bound during computationally demanding phases of the simulation, and use for the rest of the simulation a lower delay value.

The simulator should issue a warning when the user decides not to specify the value T, and let the delay introduced by the physical device not controlled by the simulator and time-varying. Moreover, an additional warning should be issued when data must be injected into more than one component, and the time of the data delivery to them is not the same. This occurs in a scenario like Figure 9-5 when the user decides not to introduce the delay module in the model.

9.3 Implementation using HLA

A physical device in the SPRINT simulation platform is represented as a HLA federate. The PD proxy is implemented using the HLA services, and it is embedded in the HLA interface module of a physical device (Figure 6-6). We will summarize next the synchronization mechanism in HLA before presenting how physical devices coordinate with simulators over HLA.

9.3.1 Conservative time synchronization mechanism in HLA

In the HLA jargon, the simulation actors that participate in a distributed (HiL) simulation create a *federation*; each simulation actor joining a federation is a *federate*. Each federate has an internal notion of time.

Federate exchange data during a simulation. There are two kinds of messages that can be used for communication:

- Receive Order (RO) messages: these messages are received by a federate as soon as they are transmitted.

- Time Stamp Order (TSO) messages: these messages are sent with an attached timestamp. A federate receives them when its internal time reaches the timestamp value.

HLA provides services to synchronize the internal time of federates. With regard to time synchronization, a federate may be:

- Time regulating, when its internal time is free to progress but it is used by other federates as a reference to synchronize their own internal clock.
- Time constrained, when the internal time of the federate progresses accordingly with the time of the time regulating federates.

A federate may be regulating, constrained, both regulating and constrained, neither regulating and constrained. Time Stamp Order messages can be sent by time regulating federates and received by time constrained federates. Federates which are not regulating can only send RO messages, whereas not constrained federates will receive all data in the form of RO messages (TSO messages are delivered to not constrained federates as RO messages).

The synchronization among the federates relies on an explicit time advance request performed by the federate to the HLA coordinator. The HLA coordinator is responsible for granting the time advance (time advance grant). Federates which are not time constrained are always granted to advance their time. Time constrained federates can advance according to the status of the time regulating federates. More in detail, the HLA coordinator guarantees that, when granting a time advance to time T, no TSO messages will be generated with timestamp less than T by the time regulating federates, i.e. TSO messages will always arrive to a constrained federate in chronological order and no TSO message will be received by a federate with a timestamp in the past with respect to its internal time. Notice that the HLA coordinator delivers TSO messages to a federate just before issuing a time advance grant.

Time regulating federates need to declare a lookahead value. The time regulating federate cannot generate TSO messages with a timestamp less than its current time plus the lookahead value. A lookahead larger than zero allows a constrained federate to move in the future with respect to the current time of the regulating federate, preserving the guarantee that TSO are received in chronological order. Lookahead can have a dramatic effect on the speed of the simulation [21].

9.3.2 Physical devices as time regulating and constrained

Messages exchanged between the PD proxy and the other federates are required to carry timing information for proper processing on both sides; we hence resort to HLA TSO messages for communication among the federates. This implies that all the federates act in general as time regulating and constrained. HLA time synchronization services are used to guarantee that all the federates have a consistent notion of time, so as to guarantee that the virtual time t_v has the same value all over the system model (as assumed in section 9.2.1).

The federate associated with a physical device will synchronize its virtual time (t_v) with the other federates and with the real time of the device (t_r). Figure 9-6 shows the two synchronization mechanisms used by a federate associated to a physical device. The federate tries to increase its time at the same rate of a local wall-clock time source, whereas the HLA services guarantee that time progresses consistently with the other federates. As a result, the physical device may be prevented from running at real time from the other federates, and the other federates cannot run faster than real time.

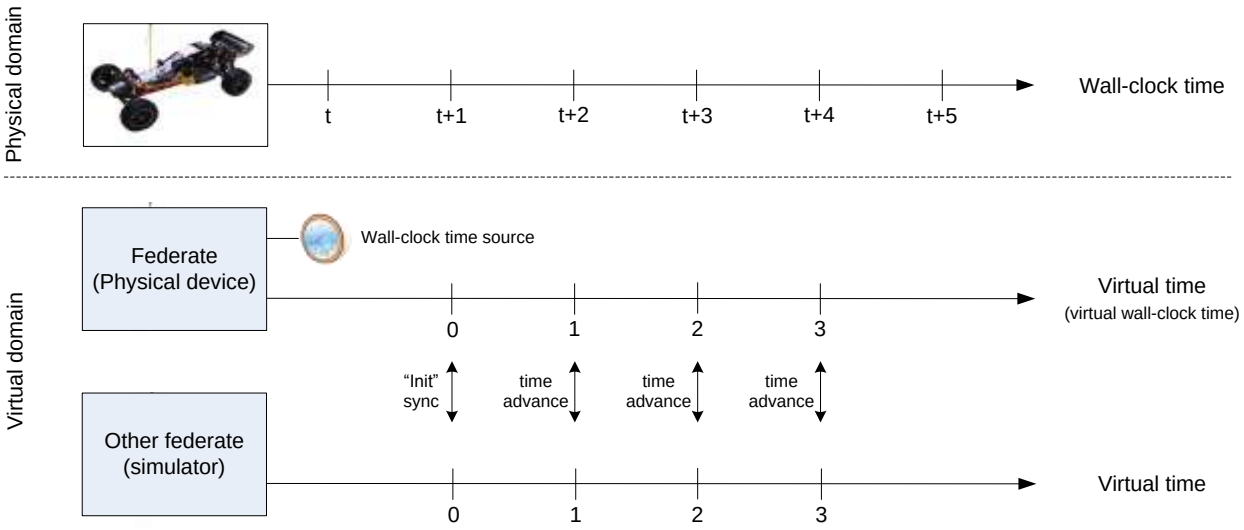

Figure 9-6: Synchronization of federates using HLA services

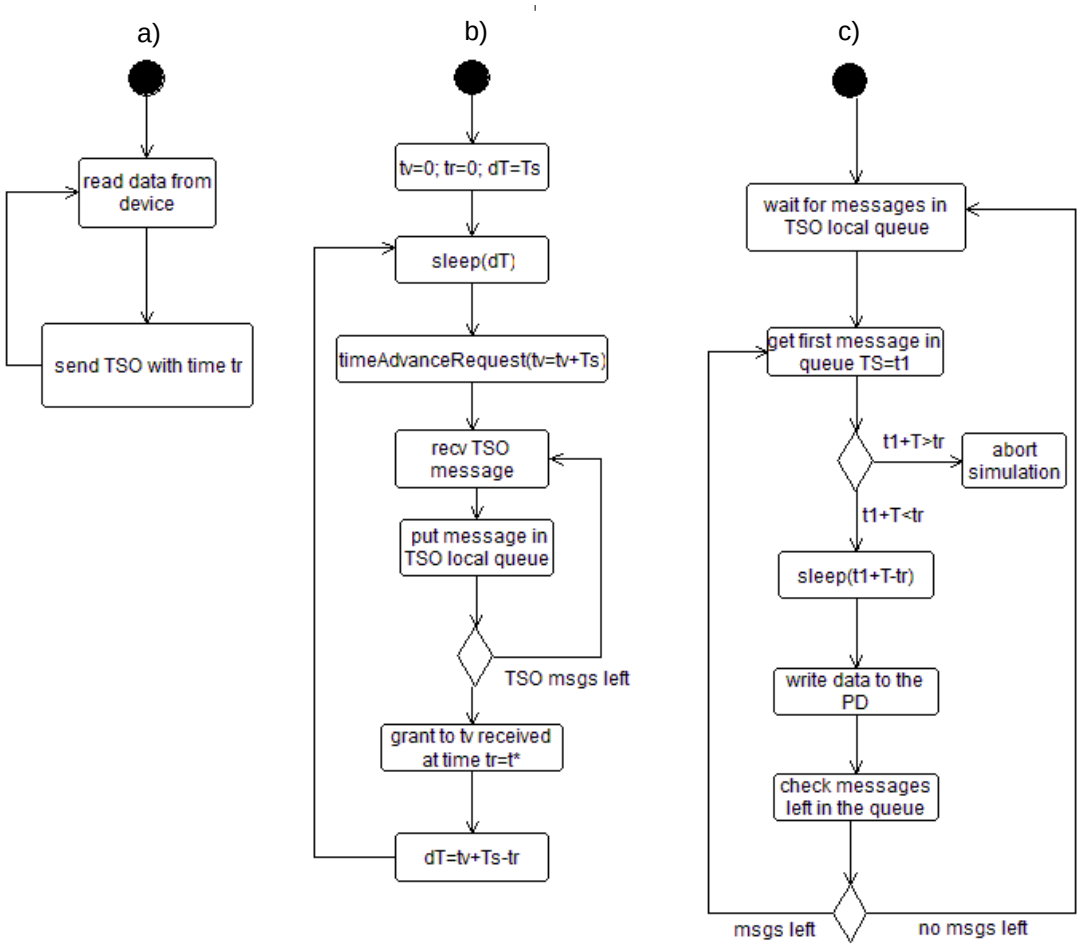

Figure 9-7: Behaviour of a physical device federate.

Figure 9-7 shows the behaviour of the physical device federate. The federate manages two notions of time: the wall-clock time (t_r), which is not under the control of the federate, and a virtual clock (t_v). The behaviour of the federate is described by three threads, dealing with:

a) The reading of data from the physical device and their transmission as TSO messages;
b) The time synchronization and the reception of TSO messages
c) The writing of the received data from the other federates to the physical device at the right point in time

The synchronization/reception thread starts with the initialization of t_r and t_v. T is the period of the time advance request, whereas dT is the amount of time the federate sleeps before requesting a time advance. At the first iteration, dT=T. Just after the sleep, t_r is equal to T_s, and the federate requires to advance t_v to T_s; we remark that the objective of the federate is to maintain t_v synchronized with t_r (ideally, t_v always equal to t_r).

Just before the time advance grant, TSO messages are delivered to the federate. The received data are stored in a local queue in the order they are received; HLA guarantees that messages are received in timestamp order.

The federate is granted to advance to T_s; at this point, t_r has increased its value of the time required for the time advance grant procedure. The federate tries to compensate the difference between t_v and t_r by adjusting the amount of time to sleep (dT) before requesting a new time advance of an amount T. Notice that, except for the first iteration, T will always be larger than dT.

In parallel to the synchronization and reception of messages, the transmission occurs. As soon as data are read from the device, they are sent using TSO messages. The timestamp of the message is the current value of t_r. Notice that this is legal because t_v is always smaller than t_r, and a federate is allowed to generate messages with timestamps in the future.

The last thread waits for the presence of messages in the local queue. As soon as a message is pushed in the queue, it gets the timestamp of the message (whose value is supposed to be t1 in the diagram), and evaluates if time t_1+T (the delay value specified by the user) is in the future with respect to t_r. In case it is, this is a violation of the bound defined by the user, and the simulation is aborted. Otherwise, the thread sleeps up to the point where tr is equal to t_1+T and then writes the data in the message to the device. Then, it processes other messages in the local queue, if available, or goes back to waiting for the arrival of other messages.

9.3.3 Physical devices as time regulating only

Physical devices that have just output interfaces (e.g., sensors) do not need to receive (TSO) messages from the other federates. They can hence act as time regulating only. The federate behaviour is as in Figure 9-7, but no messages are received before the grant, hence thread c is never executed. Moreover, the grant is issued regardless of the state of the other federates; therefore, the virtual time in the physical device is free to progress at the wall-clock time rate. All the other constrained federates use the time of the federate as reference, and this prevents them from eventually running faster than real-time.

9.3.4 Physical devices as time constrained

Physical devices that have just input interfaces (e.g., actuators) do not need to send (TSO) messages to other federates. They can hence act as time constrained only. There is no need for time regulation in this scenario:

- If the other federates execute faster than the physical device, they will generate messages with timestamp in the future with respect to the wall-clock time; therefore, these messages can be delivered to the device at the right point in time.
- If the other federates execute slower than the physical device, the time regulating mechanism has no way to speed them up.

The behaviour of the federate is as in Figure 9-7, but the sleep can be moved after the reception of the grant. Basically, the federate first increases the federate time and then wait for the wall-clock time to get aligned with it. In this scenario, t_v may be larger than t_r, and TSO messages may hence arrive with a timestamp that is still in the future with respect to t_r. Nevertheless, it is still needed to introduce an extra delay T in the message timestamp to guarantee their delivery at a pre-defined point in time.

9.4 Mitigation of real-time synchronization flaws

The proposed synchronization approach tries to preserve the timing relation of events between the physical devices and the models with the introduction of an extra delay that compensates the de-synchronization between the virtual time and the real time. Nevertheless, this delay affects the simulated trace. In this section, we will describe application scenarios of HiL simulation where the constraint on real-time synchronization is less stringent, together with techniques to reduce the de-synchronization error.

9.4.1 Application scenarios

Application scenarios exist that are not affected by the imprecise synchronization between the physical devices and the simulators. When physical devices have just output interfaces (as in section 9.3.3), they can run ahead in time; data generated on their output interfaces can be injected in the simulator at the proper point in time.

Particular classes of devices are stuttering-invariant [14]. Their state does not change over time, but just in reaction to an input event. This implies that the state and output trace of the device generated by an input sequence does not change when some of the inputs are delayed or anticipated, but the ordering of the inputs is preserved. In the presence of this kind of devices, the delay introduced in the interaction with the physical device does not modify the physical device execution.

Stallable devices are a particular class of stuttering-invariant devices; their execution can be suspended and resumed without affecting their behaviour. In this scenario, the PD proxy can control the execution of the device to minimize the de-synchronization between the real time and the wall clock time. The synchronization thread (thread b in Figure 9-7) could resume the device execution just before the sleep period and suspend the device execution right after it, so that the time of the device does not progress while waiting for a time advance grant.

9.4.2 Simulation techniques

There are two major factors that may prevent a real-time execution of a HiL simulation:

- The computational complexity of the simulation models, that limit the speed at which simulated time can progress
- The communication delays.

In this deliverable we will not discuss techniques to speed up the simulation, because they are simulator-specific. However, some recommendations may be provided:

- Use of abstract simulation model for HiL simulation has to be considered. Refined model might not provide more accurate results considering the delays introduced in the interaction with the physical devices.
- The distributed simulation architecture may be exploited to partition complex models and distribute them on different computational resources.
- Regulating federates might use a lookahead larger than 0 to speed up the overall simulation. In particular, a not null lookahead adopted by the physical device federate, when acting as regulating, allows simulators to move in the future with respect to the time of the physical device, of an amount of time given by the lookahead value. The lookahead can hence compensate temporary slow-down of the simulator. Notice that the use of the lookahead requires the physical device federate not to generate messages with timestamp less that its virtual time t_v plus the lookahead value.

SPRINT envisions the communication among tools, models, physical devices and engineers over the Internet (“Internet of System Engineering”). Therefore, the communication link among the federates is not a design choice. The reduction of the communication delays is expected by the availability of communication services with QoS guarantees, which are currently available in the service portfolio of a few Internet providers [25], and by a general improvement of the Internet infrastructure, led by the increase of applications demanding low latencies communications (e.g., VoIP, video streaming).

10 Example

Figure 10-1 shows a system that consists of a joystick, a Low Level Controller and a RC car. The LLC transforms the output from the joystick to suitable control signals for the RC car.

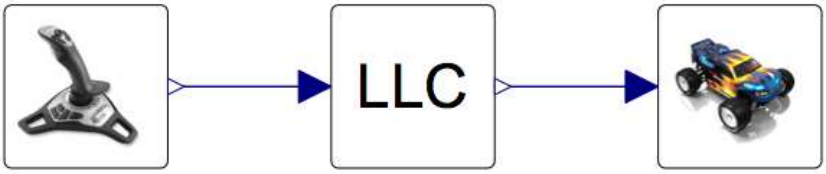

Figure 10-1: System model.

Now assume the systems engineer needs to simulate the system, we assume the following

- Joystick exist as a physical device
- LLC model in Rhapsody
- RC Car model in SystemModeler
- RC Car exists as a physical device

The scenario is as follows, the Rhapsody implementation of the LLC will be used to feed the joystick signals to both the SystemModeler model of the car and the physical car. An example deployment for this scenario is shown in Figure 10-2. To achieve that the engineer will use the dashboard to

1. Create a simulation specification, selecting the implementation or physical device that should be used for each component. As a part of the process a suitable simulator has to be chosen for each virtual implementation
2. Start the simulation, the simulation dashboard will then dispatch the component execution to their respective simulator or physical device.
3. Each simulator will then fetch the implementation from SII and wait for the simulation dashboard to start the simulation.
4. When the simulation has finished each simulator uploads the result for its component to the SII.
5. The engineer can fetch the results from the SII and analyze them.

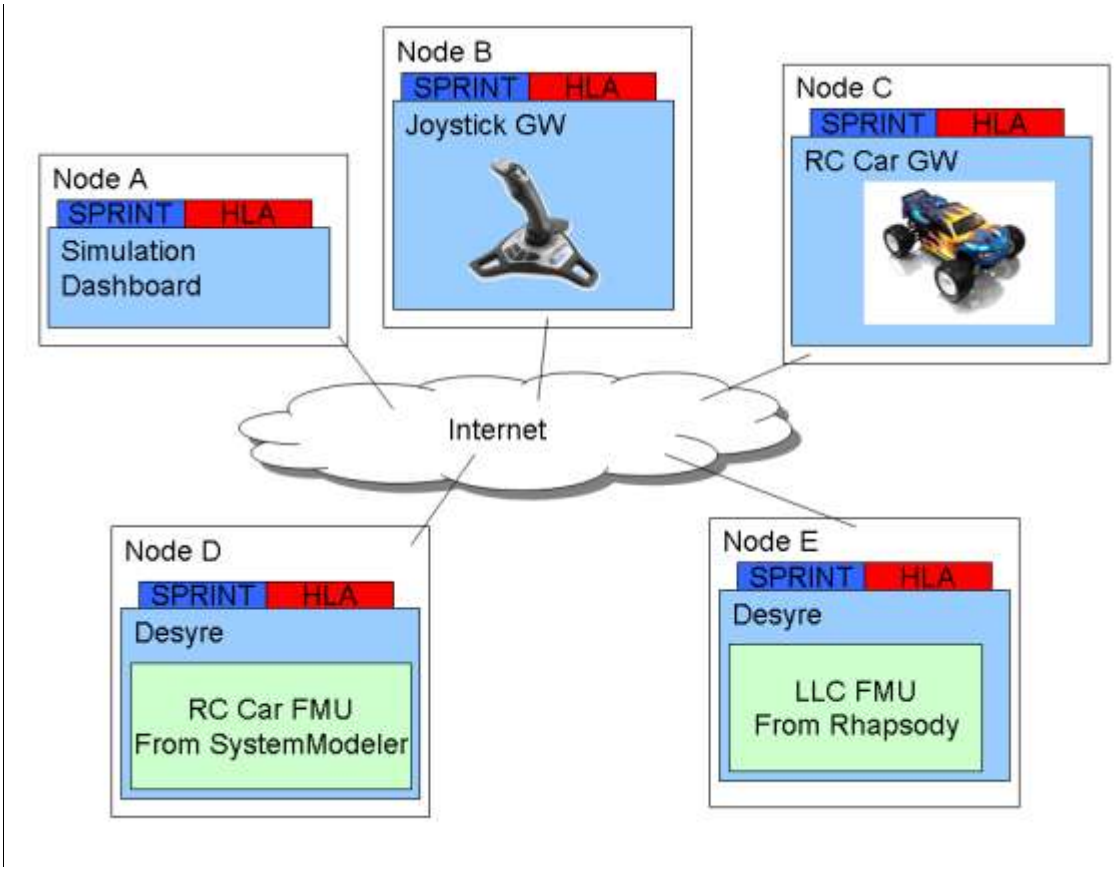

Figure 10-2: Deployment scenario.

11 Performance

This section aims at capturing a set of initial considerations on the performance of the distributed HiL simulation platform, in terms of the attainable rate between the wall-clock time and the simulated time. A rate larger than 1 implies that the simulated time progresses slower than the wall-clock time. For instance, a rate of 2 means that to simulate 1 minute of system activity, 2 minutes of wall-clock time are needed.

It is important to stress that the impact of performance is twofold:

- Low simulation performance may affect the simulation usability. Users will not adopt distributed simulation if simulation times are much higher than in a non-distributed scenario.
- When physical devices are involved in the simulation, the ratio between wall-clock time and simulated time is required to be approximately equal to 1. Physical devices run in real-time, and the synchronization of the simulated part with the physical part imposes a real-time constraint on the whole system model execution. Low performance may prevent the simulation from meeting this constraint, therefore limiting the usage of physical devices in simulation.

Performances are affected by different factors. Some of them are not specific to distributed simulation, but apply instead to simulation in general. The complexity of the model to be executed, for instance, has a major impact on performance. In this section, we will focus on the performance penalties that incur in distributed simulation. There are two major contributions to be considered:

- The communication overhead.
- The coordination overhead (HLA overhead)

They will be analysed in the following subsections.

11.1 Communication overhead

The actors involved in the distributed HiL simulation communicate over the public Internet. Internet provides (in general) a best effort communication service, with no guarantees in terms of QoS. In particular, the latency of the communication is expected to have the largest impact on the simulation performance.

	Link	Round-Trip Delay [ms]
Inter Region	New York to London	74.76
	New York to Rio De Janeiro	111.81
	San Francisco to Tokyo	112.02
	Singapore to Los Angeles	189.94
	Tokyo to London	252.41
Europe	Frankfurt to Milan	10.84
	Frankfurt to Stockholm	27.53
	London to Madrid	32.50
	London to Paris	8.98

Table 11-1: Round trip delays on the AT&T Global IP Network (average Feb'11 – Jan'12) [27]

Table 11-1 provides the average values of the round trip delays for several links in the period February 2011 to January 2012. The delays are measured between nodes of the transport network; therefore, they do not include the delays of the access network.

These values provide a quantitative measure of the communication overhead. Each interaction among the simulation actors over the Internet may cost, in terms of time, as much as 250 ms. Let us consider a simple scenario with a physical device and a simulator, performing HiL simulation over the Internet (Figure 11-1). The physical device sends a message to the simulated model and gets back another message after some elaboration of the model. This scenario fits, for instance, with a model of a control system, where the physical device is a controller and the simulated model captures the actuators, the plant and the sensors. The controller sends a control message to the actuators, the model performs an execution step to calculate the state of the plant after the actuation, and a sensor sends a message to the controller with the new state after the actuation. As shown in the picture, the whole interaction loop has a delay equal to the round-trip delay

plus the model computation delay. Even if we assume that the model computation takes 0 time, we still have that the whole loop may require as long as 250 ms. Since the physical device cannot be suspended, the communication delay affects its execution. It prevents, for instance, the simulation of a controller that requires more than 4 control loops in a second.

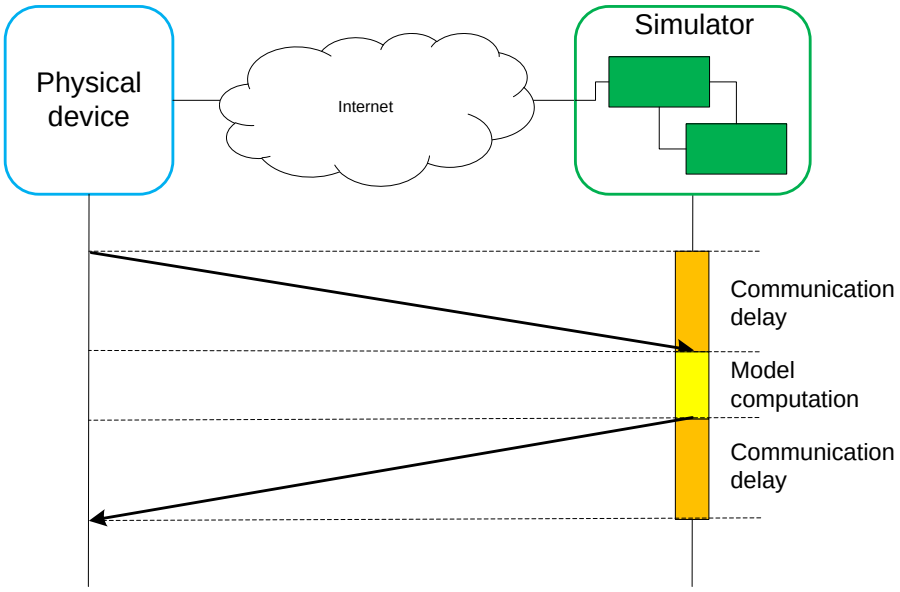

Figure 11-1: Interaction between a physical device and a simulator over the Internet

There are a few considerations that can be done on the impact of Internet latencies on the simulation execution. First of all, the value of 250 ms is a worst case scenario. Communications inside Europe, for instance, are much faster, and they would hence support quicker interactions.

Moreover, the statistics in Table 11-1 show the current performance of the network. In the future, it is likely that the quality of service over Internet will improve. New applications like Voice-over-IP and video conferences demand for bounded delays, and technologies for QoS management on IP network (e.g., DiffServ) are already available to serve these requests. Internet access with QoS guarantees is currently available in the service portfolio of a few internet providers [25].

As already pointed out in section 9, the class of stuttering invariant devices does not require to be executed in real-time, and it is hence insensitive to network delays. For these devices, distributed HiL simulation can surely be performed over the Internet, even without any QoS guarantee.

11.2 Coordination overhead

Coordination among the simulation actors participating in a distributed HiL simulation is performed by the HLA platform. Coordination involves synchronizing the execution of the different federates, and guaranteeing that events generated in different simulators or physical devices are processed in a global chronological order. These functionalities have a cost in terms of performance of the simulation execution.

To achieve a first assessment of the coordination overhead, we have performed some measurements over a simple model. The model is composed by logical units modelling a clock; the clock has a parametric period, which may be different for each instance of the clock in the model. Each unit is composed by two threads (Figure 11-2). The first thread sends to the other nodes the time it will reach in the next tick. The message carries a timestamp, stating when the message must be processed, and a time value (set equal to the timestamp). After sending the message, the clock increases its time and prints the value of its current time. The second thread receives the messages from the other clocks, and prints them. The activity of a clock ends when its time exceeds the simulation time. In a scenario with multiple clocks running in parallel, each one progressing in time according to its own period T, the synchronization among them guarantees that the printouts generated in a clock by the two threads, representing the local time and the time values received by the other clocks, are chronologically ordered. Thread 2 will not receive and print messages that are in the past with respect to the local time, and thread 1 will not print time values in the past with respect to the messages received and printed by thread 2.

Two implementations of the model just described have been defined:

- **Implementation 1**: pure SystemC implementation. The model is not distributed and all the threads are sequentially executed (no parallelism). Each clock is implemented by a sc_module, and clock modules communicate through sc_ports. Synchronization among the clock modules is guaranteed by the SystemC simulation kernel.
- **Implementation 2**: SystemC + HLA implementation. The model can be distributed, and clock modules can run in parallel. Each clock runs in a different SystemC simulator, acting as a federate in a HLA federation. Communication among the clocks is performed using TSO interactions. The SystemC simulators are synchronized using the HLA event-based synchronization mechanisms (*NextEventRequest* primitive). The CERTI HLA implementation [18] is used (available in public domain).

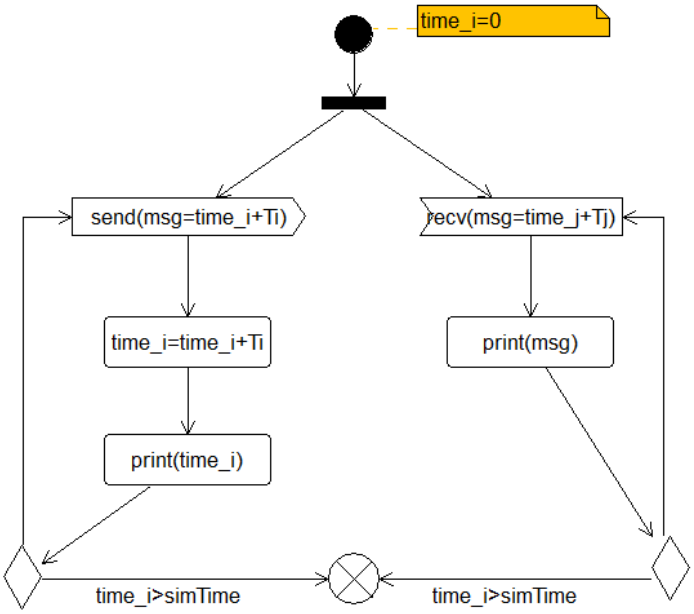

Figure 11-2: Activity diagram of the simulation model used for assessment of HLA overhead

Notice that each clock module performs almost no computation, but requires frequent communication and coordination with the other modules. We hence expect this test to estimate a worst-case overhead for distribution of the simulation on a HLA platform.

The test is performed on a laptop with an Intel Core I7 Q840 (4 cores, 8 threads). The executed model is composed by three clocks, with periods 2 s, 3 s, 5 s. The simulation time is set to 864000 s (10 days). Implementation 1 performs the simulation in about 7 s, whereas implementation 2 requires 1330 s. The overhead of the HLA synchronization is of about two orders of magnitude (190 x), but still guarantees a ratio between simulation time and simulated time smaller than one. Notice that the clock modules in implementation 2 are all running on the same hosting machine, but parallelism allows exploiting the multi-core architecture of the CPU. There is hence no communication going over a network, and hence no extra overhead due to network delays.

In terms of performance, distributed simulation can be a winning solution when the model to be executed is computationally intensive, so that splitting the computation load on different machines can really impact the execution times. To test this scenario, we have modified the model to increase the computation performed by each clock module. Between the sending of the message and the increase of the local time, thread 1 increases an integer variable from 0 to 1E8 with one-by-one steps inside a loop. The simulation time is set to 3600 s (1 hour). Implementation 1 performs the simulation in 17 s, implementation 2 in 620 s. The overhead drops from 190 x to 36 x.

12 Simulation platform and IoT

This section highlights the concepts in common between the Internet-of-Things domain and the architecture of the simulation platform, and envisions the usage of simulation in the IoT design flow.

12.1 IoT concepts in the simulation platform

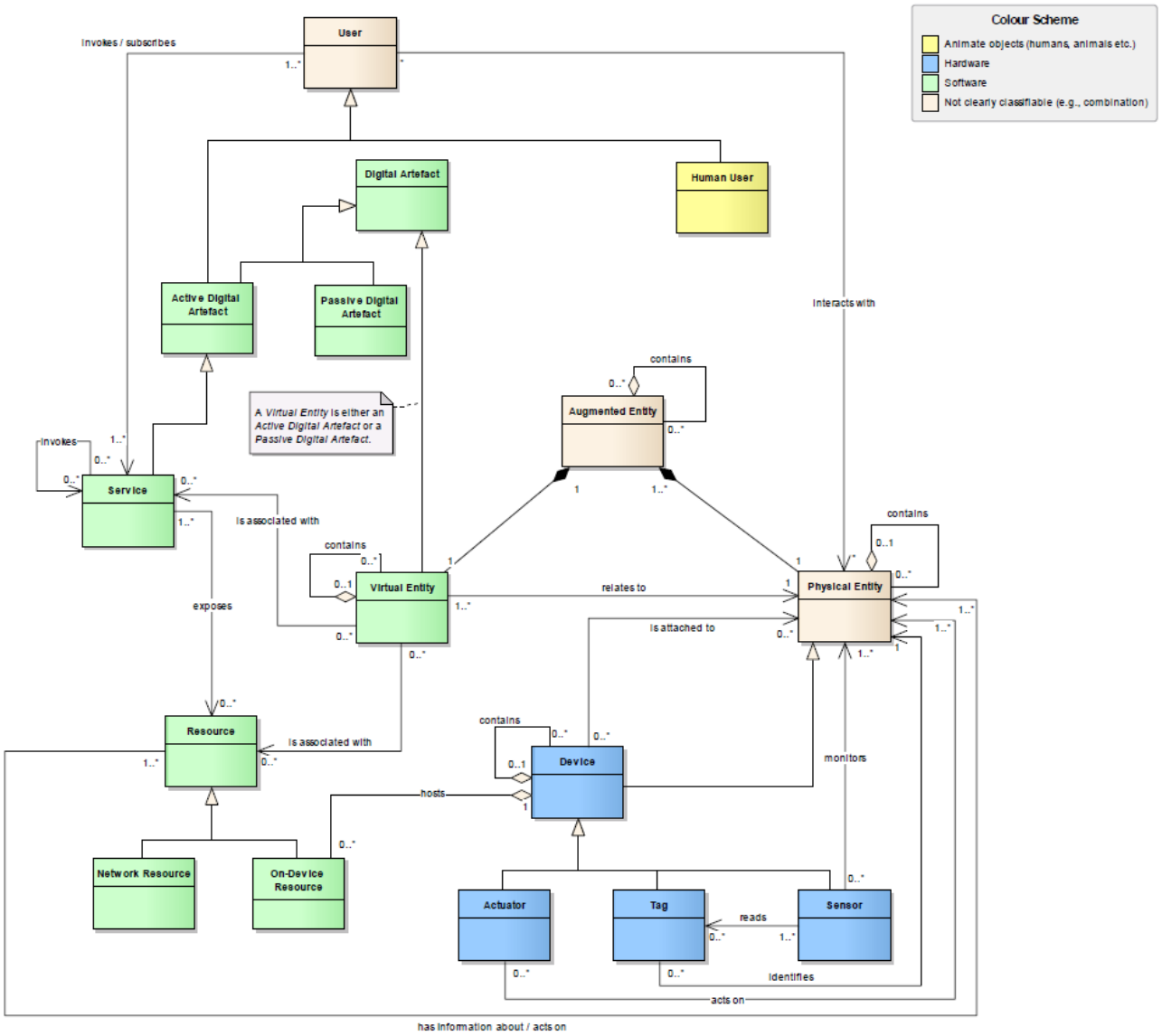

Figure 12-1: Reference model of IoT (IoT-A project)

Figure 12-1 shows the fundamental concepts of an IoT system and their relations. This reference model has been defined in the context of the IoT-A project [28], which aims at producing a unified vision of the IoT among the different EU projects working on the topic. SPRINT is one of them, and this motivates the need to compare our architecture with the abstraction provided by the IoT-A researchers. More in detail, in this section we will focus on the simulation platform architecture and we will try to identify points of contacts and eventually discrepancies with the IoT reference model.

The physical devices participating in a HiL simulation are physical entities in IoT-A. The notion of physical entity includes devices (as explicitly stated in the model), but it is intended to be wider, including any physical object.

Each physical entity is associated with 1 or more virtual entity. In the SPRINT simulation, a device has two virtual representations:

- A record in the SPRINT platform, which represents the device as a model element and enables users to discover its availability and features. This representation is classified as a passive digital artefact in the IoT-A jargon.
- A HLA federate, which allows the device to participate in a distributed simulation. This representation is an active digital artefact, since it invokes services exposed by resources to perform the simulation.

The projection of the device in the virtual domain is performed by the SPRINT gateway, the HLA coordinator and the SPRINT platform, which are hence devices and resources in the IoT-A jargon. More in detail, the SPRINT platform and the HLA coordinator may be classified as on-line resources; regarding the SPRINT gateway, the software components implementing the SPRINT, service and HLA interfaces are on-device resources hosted by the actual physical entity or an additional device interfaced with it (as shown in Figure 6-6). We remind the reader that in Figure 6-7 we have envisioned the use of RFIDs for the registration of devices, as it is typically done in IoT systems to create virtual counterparts of physical entities.

The SPRINT “resources” expose services that enable the creation of virtual entities and to interact with them. The SPRINT platform enables the discovery of the available devices and their features, whereas simulators and physical devices expose services to start the simulation. These services are mainly accessed by human users, which select the component implementation to define a simulation specification and then start the simulation. These operations are performed through the dashboard. The HLA coordinator provides services for communication and synchronization among the simulation actors. These services are not directly invoked by the simulation user, but by the active digital artifacts that represent the actors (simulators and physical devices) involved in the simulation as HLA federates.

An extended notion of physical entity would allow accommodating in the reference model other components of the SPRINT simulation platform. Models used in simulation are not physical objects, but do not natively have a virtual representation in the cloud; in SPRINT, they are related to the same kind of virtual entities used for physical devices: they are registered on the SPRINT platform and they are represented as HLA federates. Models do not require a SPRINT gateway but a simulator to be part of the federation; simulators may hence be considered as additional IoT resources.

Table 12-1 summarizes the relations between the SPRINT simulation platform and the IoT-A reference model.

Table 12-1: Mapping of SPRINT simulation platform elements on IoT-A reference model

SPRINT simulation platform	**IoT-A reference model**
Physical device	Physical entity (device)
Model	Physical entity
HLA federate	Virtual entity (Active digital artefact)
Physical device/model entry in the SPRINT platform	Virtual entity (Passive digital artefact)
SPRINT platform	Network resource
HLA coordinator	Network resource
HLA interface, SPRINT interface, service interface	On-device resource
Simulator	Resource
SPRINT gateway	Device
Discovery of available model and devices	Service (User: Human user)
Simulation	Service (User: Human user)
Distributed simulation coordination	Service (User: HLA federate)

12.2 HiL simulation for IoT design

In the previous section we have shown that the structure of the SPRINT simulation platform is consistent with the IoT-A reference model; physical devices (and models) are managed like “things” of a IoT architecture.

In this section, we envision to exploit this peculiarity of the simulation platform to support the design of an IoT system. The intended use case is the testing of an IoT system before its actual deployment, using the possibility to replace part of the physical objects with simulation models. Simulation can also be used to assess the compatibility of devices with existing IoT system installations.

In the following, we list the features of our simulation platform that makes it suitable for this application:

- The simulation platform manages both models and physical objects, and represents them using the same “virtual entity” abstractions.
- The resources (in the IoT-A jargon) of the simulation platform, namely the SPRINT gateway and the simulator, may be modified to generate additional “virtual” representations of things, closer to intended ones for the IoT system in operation. This follows the idea in Figure5-2, where the whole simulation platform exposes to the user the same service interface of the final system. This enhancement is favoured by the architecture of the simulation platform, where all the resources are accessible over the network as servers, and can be hence enhanced to expose additional abstractions to the user.
- The distributed architecture enables the integration of components distributed around the globe. It is unlikely that all the components of an IoT are developed in the same location, and a distributed testing infrastructure is surely a must.
- HLA can natively manage dynamic scenarios where federates, i.e. system components, enter and leave the simulation. This mimics the IoT behaviour, where “things” can enter and exit into the system. Notice that dynamicity is not typically managed by simulators, which require to pre-define the set of model elements”.

13 Abbreviations and Definitions

Abbreviation	Definition
OSLC	Open Services for Lifecycle Collaboration
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
BLC	Block-based Contract Language
HLA	High Level Architecture
RTI	Run Time Infrastructure
SSI	Semantic Services Integration
SII	Semantic Interoperable Integration
LAN	Local Area Network
HiL	Hardware-in-the-Loop
OSLC AM	OSLC Architecture Management
URI	Uniform Resource Identifier
TSO	Time Stamp Order
RO	Receive Order
IO	Input Output
ERS	Emergency Response System
C4I	Command, Control, Communications, Computers, and Intelligence
LIN	Local Interconnect Network
RPM	Rotation-Per-Minute
QoS	Quality-of-Service
SIM	Simulator

14 References

[1] K. Al-Zoubi et al. , “Performingi distributed simulation with restful web-services,” in Winter Simulation Conference, 2009.

[2] S. Meftali et al., “SOAP Based Distributed Simulation Environment for System-on-Chip (SoC) Design,” in Forum on Specification and Design Languages, 2005.

[3] A. Usov et al., “The DIESIS Approach to Semantically Interoperable Federated Critical Infrastructure Simulation,” in 2nd International Conference on Advances in System Simulation, 2010.

[4] C. Boer et al., “Distributed simulation in industry - A survey Part 1 - The COTS vendors,” in Winter Simulation Conference, 2006.

[5] S. Strassburger et al., “Future trends in distributed simulation and distributed virtual environments: results of a peer study,” in Winter Simulation Conference, 2008.

[6] F. Kuhl et al., Creating Computer Simulation Systems - An Introduction to the High Level Architecture, Prentice Hall, 1999.

[7] “EODiSP,” [Online]. Available: http://www.pnp-software.com/eodisp/.

[8] T. Ersal et al., "Statistical Transparency Analysis in Internet-Distributed Hardware-in-the-Loop Simulation," Mechatronics, IEEE/ASME Transactions on , vol.17, no.2, pp.228-238, April 2012

[9] W. Mahnke et al., OPC Unified Architecture, Springer, 2009.

[10] OPC - From Data Access to Unified Architecture, VDE VERLAG GMBH, 2010.

[11] “FMI v2.0 beta3,” [Online]. Available: http://www.functional-mockup-interface.org/fmi.html.

[12] “Modelica newsletter 2012-1,” [Online]. Available: https://www.modelica.org/publications/newsletters/2012-1.

[13] “Tool support for FMI,” [Online]. Available: http://www.functional-mockup-interface.org/tools.html.

[14] A. Benveniste et al., “Composing heterogeneous reactive systems”, ACM Trans. Embed. Comput. Syst., August 2008

[15] “D2.2 SPRINT deliverable – Project Requirements”.

[16] “D5.7 SPRINT deliverable – Architectural principles for Internet of System Design and the IoT”

[17] “D5.3 SPRINT deliverable – High and Low level design description of SD and SII”.

[18] “CERTI Open Source HLA RTI,” [Online]. Available: http://savannah.nongnu.org/projects/certi.

[19] W3C Wev Services Architecture - http://www.w3.org/TR/ws-arch/

[20] “Architecture Management Specification Version 2.0,” Open Services for Lifecycle Collaboration, [Online]. Available: http://open-services.net/bin/view/Main/AmSpecV2.

[21] R. M. Fujimoto, “Time Management in the High Level Architecture,” Simulation, 1998.

[22] “D3.7 SPRINT deliverable – Definition of the Semantic Services Integration Layer”

[23] “Open Services for Lifecycle Collaboration (OSLC),” [Online]. Available: http://open-services.net/.

[24] Department of Defense - Defense Modeling and Simulation Office, “RTI 1.3 - Next Generation Programmer's Guide,” 2000. [Online]. Available: http://sslab.cs.nthu.edu.tw/~fppai/HLA/RTI%201.3/RTI_NG13_Programer%20Guide.pdf.

[25] AT&T, “AT&T High Speed Internet Business Edition Service Level Agreements,” [Online]. Available: http://www.att.com/gen/general?pid=6622.

[26] "SPEEDS_Hosted_Simulation_API_definition.pdf," [Online]. Available: http://speeds.eu.com/downloads/SPEEDS_Hosted_Simulation_API_definition.pdf.

[27] AT&T, "Global network latency averages," [Online]. Available: http://ipnetwork.bgtmo.ip.att.net/pws/global_network_avgs.html.

[28] IoT-A project - http://www.iot-a.eu

[29] "D4.10 SPRINT deliverable – Contract/Sensor/Actuator integration services definition"

[30] "FMI v2.0 beta3," [Online]. Available: http://www.functional-mockup-interface.org/fmi.html.