The High Level Architecture RTI as a master to the Functional Mock-up Interface components

Muhammad Usman Awais*, Peter Palensky†, Atiyah Elsheikh‡ , Edmund Widl§ , Stifter Matthias ¶
Emails: (*Muhammad.Awais.fl, †Peter.Palensky, ‡Atiyah.Elsheikh, §Edmund.Widl ,¶Matthias.Stifter)@ait.ac.at
Austrian Institute of Technology, Vienna, Austria.

***Abstract*—Recently many commercial and non-commercial Simulation Packages (SPs) have agreed to use the Functional Mockup Interface (FMI) as the medium of interoperability. FMI presents numerous opportunities to utilize highly specialized SPs for modeling, and simulating multidisciplinary applications with several components of various types. However there is one thing missing in the FMI; the master algorithm. The paper proposes to use, the High Level Architecture (HLA) compliant Run Time Infrastructure (RTI), as a master for the FMI compatible simulation components. The ultimate goal is to provide a completely generic and standalone master for the FMI, making FMI-based simulation components usable as plug and play components, on variety of distributed environments including grids and clouds. Towards this promising goal an initial methodology is outlined.**

***Index Terms*—High Level Architecture (HLA); Functional Mockup Interface (FMI); co-simulation; distributed simulation; simulation interoperability.**

I. Introduction

Simulation is used in a variety of industrial applications as a method to prove hypotheses and to observe real world phenomena. Several domain-specific SPs for highly-specialized modeling tasks exist. Normally a single SP cannot encompass all the aspects, when there is a challenge to simulate a scenario covering many different domains. Co-simulation is one solution of the problem, which also enables modelers to reuse already developed and tested simulation solutions, as components.

Due to its importance, there have been efforts to develop standards for simulation interoperability, exchange and reuse. FMI is one such effort [1]. The FMI specifies any simulation component, according to a well defined set of function calls. An FMI compliant Simulation Package (SP) exports a simulation in form of a shared library. Exported libraries can be used in any executable using a "C" API. The set of functions within the API is used for setting up, accessing, modifying and manipulating the simulation and its configurations. Many vendors have agreed to use FMI and now provide the facility of exporting simulations as reusable shared components [2].

The HLA [3] was developed by the U.S. Modeling and Simulation Coordination Office (M&S CO) [4]. The HLA considers individual simulations at the level of processes, rather than libraries. Later sections make it clearer that the HLA provides solutions to the most common simulation interoperability problems. It facilitates a distributed environment, suitable for military simulations. It provides a specification for the development of simulation components, such that these components remain usable even after the changes in the data model of the over all simulation. This data model is called the Federation Object Model(FOM).

Although developed differently HLA and FMI have many similarities, the paper points out these similarities and suggests a framework to use the HLA and the FMI together. The HLA RTI is proposed to be used as a master to FMI components called; Functional Mock-up Units (FMUs). Most importantly, the FMUs can be embedded components, part of a larger Cyber Physical System [2]. The proposed architecture is especially suitable in simulation of Cyber Physical Energy Systems (CPES). An in depth analysis of the problems related to simulation of CPES can be seen in [5]. Due to the broad scope of CPES, interoperability of different simulation packages becomes even more vital, for details, please see [6].

II. Standards for Simulation Interoperability

The current section presents some important simulation interoperability standards, also describing a standard reference model to examine the capabilities of these standards.

A. Interoperability Standard

The modeling and simulation community has presented many different standards for simulation interoperability, targeting the same problem from different perspectives. Some important solutions are enumerated below.

Before suggesting the HLA, M&S CO developed the Distributed Interactive Simulation (DIS) standard [7]. The DIS was specifically designed for military simulations, hence the notion of a generic co-simulation framework was missing. During its development the MITRE corporation was developing its own standard named, the Aggregate Level Simulation Protocol (ALSP) [8]. The ALSP was addressing the problem in a generic way, but many improvements were possible, which were introduced in the HLA. For example; the possibility of using many different timing algorithms in the HLA is one of its key advantages over the ALSP or the DIS. In contrast to the HLA, the ALSP was strictly distributed, without any concept of a central node and the RTI.

A web based framework named the Extensible Modeling and Simulation Framework (XMSF) [9] uses the DIS as its foundation. The XMSF addresses the issue of creating a web-based interoperable platform for simulations. Few years back the M&S CO had proposed another architecture named the

978-1-4673-5288-8/13/$31.00 ©2013 IEEE

Test and Training Enabling Architecture (TENA) [10]. TENA is not yet available open source.

Researchers working on topics related to weather and climate modeling have also faced the same difficulties in sharing each others experiences, so they have proposed solutions for their own spectrum of problems. Few of these solutions are listed here.

The Model Coupling Toolkit (MCT) [11] is a Fortran 90 toolkit for exchanging earth models. It targets multiprocessor computers and clusters. It is similar in architecture to the message passing interface. The Earth System Modeling Framework (ESMF) [12] is one of the biggest initiatives taken for modeling earth systems. It envisions coupling many different models into one entity. The Open Modeling Interface or OpenMI [13] was initiated as a model coupling platform for earth systems, but now it is further refining itself to become a generic interoperability solution for simulations.

B. Interoperability Reference Models (IRM)

The Product Development Group (PDG), working under Simulation Interoperability and Standardized Organization (SISO) has suggested the standard Interoperability Reference Models (IRM) for simulation packages [14].

TABLE I
INTEROPERABILITY REFERENCE MODELS

Interoperability Reference Models			
Category	Name	Description	Example
A	Entity Transfer	When one entity leaves a model and enters another one.	In assembly lines, one item leaves a process and enters another.
B	Shared Resource	When two models depend on a common resource. The change in the resource must take effect in both models synchronously.	A machine producing an output used by two different models.
C	Shared Event	When one event affects at more than one models simultaneously.	An alarm or signal which triggers many processes to start working.
D	Shared Data Structure	A global variable of any type used by different models.	Similar to the shared resource but handled at data structure level.

These IRMs provide a structured way to compare the capabilities of any interoperability or co-simulation standard. Table I shows the summary of these IRMs. For detailed explanation reader is directed to [14]. Keeping these IRMs in mind, the two standards (HLA and FMI) can be examined in depth.

III. THE HIGH LEVEL ARCHITECTURE

In the HLA terminology, a distributed simulation is called a "federation". A federation comprises of several components called "federates". The HLA was designed at a level independent of any language and platform. Hence it may also be considered as a protocol. For the detailed specification of the HLA, reader is directed to [3]. In the forth coming discussion the HLA is examined at the functional level and not at specification level.

The Run Time Infrastructure (RTI) plays a key role in the HLA implementation. The RTI regulates individual federates. It is the central point for communication, time synchronization, event passing and data exchange. All the communication among federates must take place using the RTI. Any federate taking an active part in HLA federation should advance in simulation time only when permitted by the RTI. It should also obey all the commands issued by the RTI, for example, state updates and event handling commands.

Sufficient understanding of working of the RTI is required to completely comprehend the next coming discussion. Reader may consult [15] and [3] for a deeper understanding. Important services of the RTI, directly related to this discussion, are mentioned below

1) Declaration management: These services are related to publishing and subscription of objects and attributes. Declarations of all the objects and events are made inside FOM. All federates must publish their global instances and attributes. Interested federates can subscribe to attributes of any instance.

2) Object management: The RTI propagates the update of an object or an attribute to all subscribing federates.

3) Ownership management: At one given time any attribute should belong to one and only one federate, otherwise there are conflicting updates.

4) Time management: These services regulate the advances in federation time. The RTI supports many different synchronization approaches. Conservative time synchronization approach guaranties that state updates or events are passed to all federates in non-decreasing time order.

5) Event management: Messages can be passed amongst federates in the form of events. If conservative approach is used, then RTI guarantees that no event arrives at the receiver later than its logical simulation time. Events can be used to propagate a condition to all the federates, for example an emergency alarm.

When comparing the HLA with respect to the SISO IRMs, the HLA supports all IRMs, except of some special cases [1]. Table II gives an overview about the IRMs which are achievable using the HLA

IV. THE FUNCTIONAL MOCK-UP INTERFACE

The FMI was suggested in the result of MODELISAR project [2]. Its main purpose was to improve the design and interoperability of systems embodying embedded software, especially used in automotive industry. Later it became popular for other types of simulation tools. The goal remained interoperability, such that a simulation component developed using one SP can be used by others, but the method was completely

[1]Further investigation of the HLA's compliance to IRMs may be required. For example the IRM category "A" has more than one special cases. One special case requires prioritization of actions. The HLA may not be able to provide direct mechanism for such prioritization. Detailed discussion is out of the current scope.

TABLE II
THE HLA AND THE INTEROPERABILITY REFERENCE MODELS

IRM		HLA Service(s)
Category	Name	
A	Entity Transfer	Ownership management
B	Shared Resource	Data distribution and declaration management with publisher subscriber model
C	Shared Event	Data distribution management with event interactions
D	Shared Data Structure	Object management using Federation Object Model

different. Instead of exporting simulations as processes, it was considered useful to export them as shared libraries.

Tools supporting the FMI should be capable of exporting FMUs. An FMU is a zip file which contains following components

- The implementation of the model in form of a library, optionally accompanying the source code.
- A model-description XML file, containing the inputs, parameters and outputs of an FMU.
- Other optional data e.g. icon images, user interface specification and relevant documentation.

One of the main objectives of the FMI is to share the mathematical simulation models, based on Ordinary Differential Equations(ODEs), Differential Algebraic Equations (DAEs) and Discrete Equations [1]. Hence the concept of numerical integrator is very important in the FMI. The FMI has two different portions of specification.

1) FMI for co-simulation
2) FMI for model exchange

In FMI for model exchange the integrator is not part of the FMU (SP importing the FMU should implement it), which makes it incomparable to the HLA. So to simplify the discussion later sections only consider FMI for co-simulation.

A. FMI for co-simulation

The concept of "master" and "slave" in the FMI is very different from that of HLA's. When an SP imports an FMU, it becomes its "master" and the loaded component becomes its "slave". FMI for co-simulation provides a mean to utilize models using an API, in a form where slave acts as a black box to master. It can react to inputs and gives outputs at discrete time steps. While using an FMU conforming to FMI for co-simulation, one does not need to know which integration method is actually applied to solve the model.

Parameters to this black box can be set in initialization phase and cannot be changed afterwards, while the inputs can be changed between discrete time steps. Functions in the FMI can be categorized in following categories. Figure 1 gives possible execution steps of an FMI component.

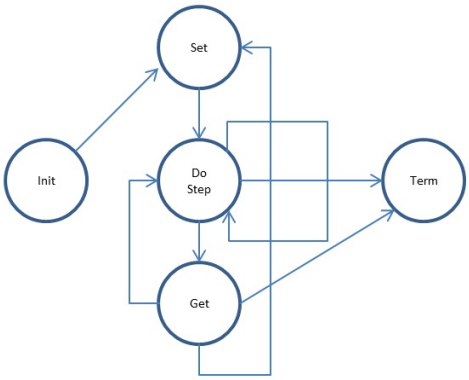
1) Initialization and instantiation functions: These are used to load the component, supply the parameters and allocate memory.

2) Progress functions: There is "fmiDoStep()" function which requests the component to step one time step ahead.

3) Getter and setter functions: Getter functions are used to read the output values and their derivatives. Setter functions can be used to set input values and their derivatives for interpolation.

4) Termination functions: These are used to unload the component and free the memory. From figure 1 it is clear

Fig. 1. States of an FMU



that there are few alternative paths of execution for an FMU, hence it is possible to write a program which loads it and performs these actions repeatedly. Next sections illustrate the idea in detail and suggest how the HLA RTI can be of use.

B. Analysis of the FMI

Analysis of the FMI according to SISO IRMs reveals that it is completely different in nature from other interoperability standards which work at the level of processes. After loading an FMU the master becomes the complete owner of the FMU and there is no question of "entity transfer". Similarly a "shared resource" does not exist in the FMI. An FMU can only read and write some values, it does not have an entity similar to a process. Naturally, there cannot be a concept of ownership management, when slave itself is owned by master.

Event generation can be identified using FMI 1.0, but it is not possible to get the timing of events [2]; there is a plan to improve event related specifications in the FMI 2.0. Sharing of data is present in the FMI to some extent, but again it is completely under the control of master, when it decides to pass the values to an FMU and reads the outputs.

The FMI is different because, FMUs are not supposed to be instantiated as standalone processes, which makes questions related to IRMs irrelevant. Nevertheless, the next subsection discusses how FMI specific simulation components can be modified, conforming them to the IRMs.

C. Elevating the FMI

Although the implementation of an FMU is contained in a shared library but it is possible to write code which can host an FMU as a process. When such an FMU interacts with other similar FMUs then the question arises, how to support IRMs for these FMUs? All the suggested functionalities of IRMs, can be provided to these FMUs by using the HLA RTI. In other words, a program which loads an FMU for simulation,

and also obeys the rules of the HLA, can act like an HLA federate. We call such a program as an "FMU-federate". A federation of such federates should naturally be called as the "FMI-federation". By obeying the HLA rules such a program guaranties that it supports all the IRMs supported by the HLA. There are many advantages of suggested technique.

- Complex FMUs generated by well-established SPs can be used directly in a federation. Integrity and abstraction of individual components (FMUs) remain intact and they are much easier to debug and troubleshoot. Each FMU can be tested in complete isolation by just providing it the desired input trajectories.
- Hosting of components on remote machines becomes much simpler. Currently any master requiring to import an FMU on a remote machine should implement a separate module for remote procedural call.
- As the RTI can support numerous different communication mechanisms, hence these FMU-federates can be hosted on any form of distributed computing architectures, including clusters, grids and clouds.
- A completely separate layer of communication allows the RTI to be shifted to any new architecture of distributed computing. If an ordinary master wishes to implement it, then it may require the vendor to change thousands of line of codes in its SP.
- Performance of a big simulation can be enhanced tremendously, by creating simulation in components and distributing them on dedicated machines.

V. Using the RTI as a master to the FMI

The current section describes the idea of using the HLA in conjunction with the FMI in its practical form, along with a proof of concept application and the results of experiment.

A. RTI as a generic master

Important services of the RTI have already been inscribed in section III. Figure 1 shows the states an FMU for co-simulation, may achieve. It is now clarified how above mentioned services can help to form an FMU into a standalone simulation process or an FMU-federate.

Time synchronization is the most important part, it enables the federates to proceed synchronously, and update federation states correctly. Secondly, in an FMI-federation there are more than one FMUs taking part, hence each FMU-federate must have mechanism to access and modify federation state variables. Third point is about events, it is more important because FMI for co-simulation does not fully support events [2]. We consider all these points one by one.

1) Declaration management: Naturally, inputs required by an FMU are the attributes it must subscribe to, and outputs are the attributes it must publish for others federates. The description of these inputs and outputs can be obtained from the model-description file. Hosting code can perform publishing and subscription generically, if some more directives are added into model-description file. These directives should guide the code about the place and role of each variable in the federation. For example code must know, which attribute in the FOM provides a certain input, so that it can subscribe to that specific attribute, and update it inside FMU whenever its value is modified. Similarly in order to publish an output variable, it must know its name and type to be assigned inside FOM.

2) Object management: The FMI does not support complex types as yet, which makes object management simpler. All the attributes of an FMI-federation can be accumulated in one FOM class, and each FMU can subscribe to interesting attributes. In this case each FMU-federate creates a new instance of the main FOM class. As it would have subscribed to some attributes, so it gets the updated states of all the instances registered so far. In other words each FMU-federate has access to the attribute values of all the instances. The number of these instances are typically be equal to the FMU-federates in the FMI-federation.

Object management code can be part of the hosting code or the FMU, depending largely on how the FOM is designed. With a single class in FOM with above mentioned scheme, object management may be made generic to a large extent.

3) Ownership management: If designers of federation can ensure that only one federate has the right to publish and update a specific attribute of one instance, then complex methods for ownership management are not needed. Otherwise, if the right has to be transfered during execution then a separate module or FMU has to be introduced. Such a module would be completely problem specific, containing policies for such scenarios.

4) Time and event management: FMI for co-simulation supports events only partially. It is advised to use the function "StepFinished()" for event driven simulation. The biggest drawback of this function is, it does not provide precise timing of an event. In the FMI 2.0 it is expected that more structures will be introduced to fully support event driven simulation, providing the proper time stamps of occurring events. Until there is no event driven simulation supported by the FMI, time management can be done only in time stepped fashion. The RTI provides the functionality by default. This may be the easiest part to implement for the hosting code, as all the precautions are taken care of by the RTI.

B. A case study

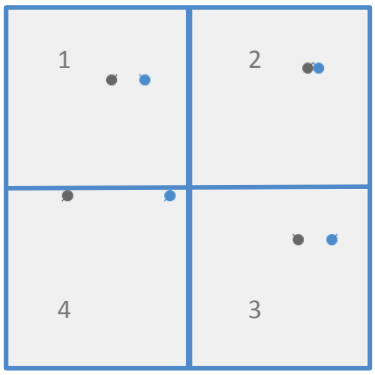
To demonstrate the proposed architecture, a simple application is developed using an open source RTI implementation, called CERTI [16]. The sample application is a federation which simulates the collision dynamics of any number of balls moving around in a square box. Each ball is simulated by an FMU. The FMU is developed using the FMU SDK [17]. The hosting code offers time, declaration and object management services from management domain. In addition hosting code has to do something related to simulation domain, due to a caveat in the FMI.

The sample application has a simple FOM, according to the suggestions made in section V-A2. It contains one class; named "Ball". It is derived from "ObjectRoot". It has two

attributes which represent the x and y position of each ball in the common frame. As each FMU-federate must subscribe to the attributes x and y so each FMU-federate is updated about the state of all other. The x and y values for FMU are random inputs in the start, but as balls collide they change the trajectories. Figure 2 shows a screenshot of a sample run. Time advances are done in time stepped fashion. The hosting code executes following steps for each time step.

1: Query the RTI about federation time
2: Request for next step and increment local time.
3: Read x and y values from the FMU
4: Enumerate reflected x and y values of other balls
5: Analyze if there is any collision
6: **if** Collision is present **then**
7: Change the direction vector of the ball
8: Update the direction vector inside the FMU
9: **end if**
10: Call FMU DoStep().
11: Read New x and y values from the FMU
12: Send update of new state to the RTI.

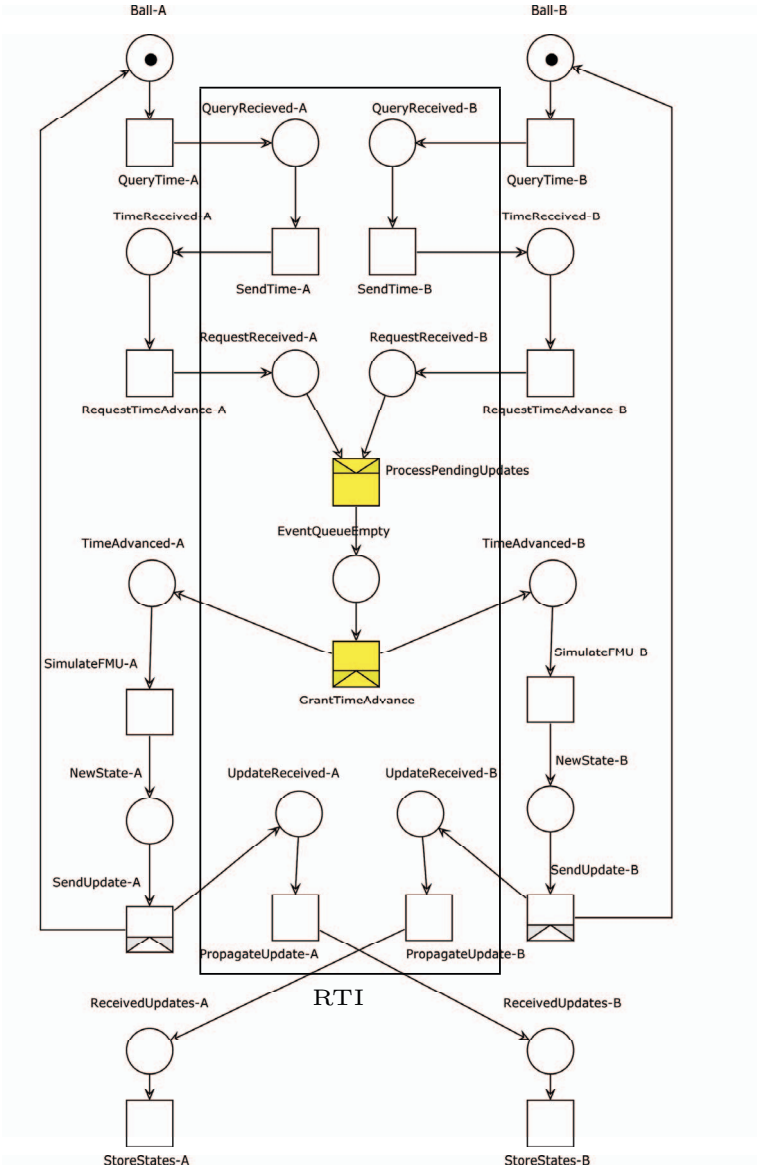
Fig. 2. Balls moving, colliding, and changing directions



The problematic part in above mentioned algorithm is steps 5 to 9, whose processing is relevant to simulation rather than management of FMU. This was necessary because there is no clear and simple mechanism in the FMI to pass arrays or lists to FMUs, hence the processing had to be done inside hosting code.

Figure 3 demonstrates the interaction of the simulating FMUs with the RTI, by the help of an informal Petri net diagram. The names of transitions correspond to the function calls or actions, while the places are the states achieved by the FMUs or the RTI. Names are chosen to be self-descriptive. Figure shows the simulation of two balls. The same concept applies for more than two balls as well. The central rectangle shows the actions and states of the RTI, while left of it are the states and actions related to ball simulated by FMU-A, FMU-B has them mentioned on the right side. The synchronization points are highlighted. The synchronization starts at transition "ProcessPendingUpdates". Any FMU reaching at this point is put to wait by the RTI until certain conditions are met. These

Fig. 3. Informal Petri net diagram of interactions of the FMUs with the RTI



conditions are mentioned below

- As sample application is a time stepped simulation so each and every FMU must proceed synchronously, hence if FMU-A has not fired the transition "RequestTimeAdvance-A" then FMU-B has to wait for it indefinitely. Same is true for more than two FMUs, all FMUs must join at this step to proceed further.
- If all the FMUs have fired the transition "RequestTimeAdvance", then before granting a time advance, the RTI must first ensure that it does not have any pending requests. These pending requests could be related to the state updates of different attributes, pertaining to the previous time step.

If above mentioned conditions are satisfied then all of the FMUs are granted the permission to proceed for next time step.

Afterwards each FMU is liberated to proceed independently. The synchronization portion ends here, highlighted in the figure as well.

The processing of each time step finishes when an FMU sends its updated state to the RTI, the rest of processing is done in a separate thread. The RTI must propagate the updated state of each FMU-federate to all the subscribing FMU-federates. Sample application has only two FMU-federates, so only one has to be updated, but if there were more, then all of them would have received the update. Step 4 in the algorithm gets the updated values in result of this. It should be clear that RTI is not a store of these values, rather it works like a messenger, which delivers the news of change in an attribute to all the subscribers.

C. Results and findings

The implementation of the above mentioned simple application shows that there are great opportunities of integrating the HLA and the FMI standards. By extending the presented concept of integration, it is evident that creation of a framework is possible. The framework should be able to host any simulation, constructed in the form of independent FMUs over different communication networks. Certainly there are some limitations, but they can be overcome. Some important findings drawn out of above mentioned experiment, are listed below.

- Time stepped simulation is the easiest configuration to form an "FMI-federation", with current state of the FMI.
- The architecture of the HLA FOM is flexible enough to be used as shared data model for proposed FMI-federation.
- The RTI provides all services , which enables it to be used as a generic master to discrete time stepped simulations.
- Event driven simulations should be properly supported in FMI for co-simulation. Specifically the time stamps for each event must be provided by an FMU.
- To a reasonable extent the hosting code for an FMU can be written generically. Following below given suggestions can improve the level of generality in the hosting code.
- Model-description file should be extended in order to make it better usable for an FMI-federation. At least, it must include the interdependencies of an FMU inputs and outputs to the attributes in the FOM. The dependencies can only be specified by FMI-federation designers. The specification can be made using optional attributes of the model-description file.
- In order to allow FMUs to work properly, it is necessary that the FMI consortium permits more optional attributes to be the part of the model-description file, by changing the schema description (.xsd) file.
- In order to make FMU-federates more robust, there should be a clear mechanism in the FMI to pass lists and arrays. Support for complex data types, can enhance the level of abstraction.

VI. Conclusion

The paper proposes the idea of using the HLA RTI as a generic master for the FMI compliant co-simulation components, broadening the scope of the FMI and bridging the gap between many established SP vendors. While developing a proof of concept application, some possible improvements in the FMI are noted. The FMI should provide support for lists and complex data types. It should also support event driven simulation. It is demonstrated that, it is possible to write structured hosts for FMUs. The host can make an FMU work like an HLA federate, hence the name "FMU-federate" is suggested. The RTI can be used to synchronize and orchestrate the FMI-federation of these FMU-federates. The proposed distributed architecture can enhance the performance, and can offer the opportunity of hosting simulations on different types of distributed computing environments, such as clusters, grids and clouds. Testing individual components in is also be easier, using proposed scheme.

References

[1] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold *et al.*, "The functional mockup interface for tool independent exchange of simulation models," in *Modelica'2011 Conference, March*, 2011, pp. 20–22.

[2] Modelisar. Consortium. [Online]. Available: http://www.modelisar.com/

[3] S. I. S. Committee *et al.*, "Ieee standard for modeling and simulation (m&s) high level architecture (hla)-ieee std 1516-2000, 1516.1-2000, 1516.2-2000. new york: Institute of electrical and electronics engineers," *Inc., New York*, 2000.

[4] Modeling and simulation coordination office. [Online]. Available: http://www.msco.mil/

[5] A. Elsheikh, E. Widl, and P. Palensky, "Simulating complex energy systems with modelica: A primary evaluation," in *2012 6th IEEE International Conference on Digital Ecosystems Technologies (DEST)*. IEEE, 2012, pp. 1–6.

[6] E. Widl, P. Palensky, and A. Elsheikh, "Evaluation of two approaches for simulating cyber-physical energy systems," in *Proceedings of the 38th IEEE Conference on Industrial Electronics IECON 2012*, Montreal, Canada.

[7] D. S. Committee *et al.*, "Ieee standard for distributed interactive simulation-application protocols," *IEEE Standard*, vol. 1278, 1998.

[8] A. Wilson and R. Weatherly, "The aggregate level simulation protocol: an evolving system," in *Proceedings of the 26th conference on Winter simulation*. Society for Computer Simulation International, 1994, pp. 781–787.

[9] K. L. M. Don Brutzman, J. Mark Pullen. (2004) Extensible modeling and simulation framework (xmsf). Consortium. [Online]. Available: https://www.movesinstitute.org/xmsf/xmsf.html

[10] U. DoD, "Tena-the test and training enabling architecture reference document," 2002.

[11] J. Larson, R. Jacob, and E. Ong, "The model coupling toolkit: A new fortran90 toolkit for building multiphysics parallel coupled models," *International Journal of High Performance Computing Applications*, vol. 19, no. 3, pp. 277–292, 2005.

[12] C. Hill, C. DeLuca, M. Suarez, A. D. Silva *et al.*, "The architecture of the earth system modeling framework," *Computing in Science & Engineering*, vol. 6, no. 1, pp. 18–28, 2004.

[13] J. Gregersen, P. Gijsbers, and S. Westen, "Openmi: Open modelling interface," *Journal of Hydroinformatics*, vol. 9, no. 3, pp. 175–191, 2007.

[14] S. I. S. Committee, "Standard for commercial-off-the-shelf simulation package interoperability reference models-siso-std-006-2010," SISO COTS Simulation Package Interoperability Product Development Group, Tech. Rep., 2010.

[15] F. Kuhl, R. Weatherly, and J. Dahmann, *Creating computer simulation systems: an introduction to the high level architecture*. Prentice Hall PTR, 1999.

[16] Certi. [Online]. Available: http://www.nongnu.org/certi/

[17] QTronic. (2010, October) Fmu sdk free software development kit. [Online]. Available: http://www.qtronic.de/en/fmusdk.html