Semantic interoperability in co-simulation: use cases and requirements

Jose Juan Hernandez and Jose Evora
University Institute of Intelligent Systems and
Numerical Applications in the Engineering (SIANI)
Edificio central del Parque Científico-Tecnológico,
Campus Universitario de Tafira, 35017
Las Palmas de Gran Canaria, Spain
jose.evora@siani.es, josejuanhernandez@siani.es

Jean-Philippe Tavella
EDF Lab Paris-Saclay
7 Boulevard Gaspard Monge
91120 Palaiseau
France
jean-philippe.tavella@edf.fr

KEYWORDS
co-simulation, semantic interoperability, FMI, domain specific language

ABSTRACT

Simulation of complex and heterogeneous systems is done by decomposing the system into many subsystems. These subsystems are individually simulated and results are integrated by a co-simulation algorithm. In this approach, an interoperability framework that allows the integration of components is required.The FMI standard is becoming the interoperability reference for co-simulation, although it is still evolving. Semantic interoperability is an interesting issue that the FMI standard could include in future versions.
Semantic interoperability is concerned with the use of explicit semantic descriptions to define how simulation components should be associated. In simulation, this issue is especially important as it provides essential support for cross-domain integration, guaranteeing consistency and composability.
In this paper, some co-simulation use cases and semantic interoperability requirements that could be considered in future versions of the standard are analysed.

Introduction

One of the challenges facing the software industry is semantic interoperability. This consists of exchanging data that is correctly interpreted by the receiving system, with the same meaning as that of the transmitting system. Semantic technologies have been used in several applications and their benefits have been demonstrated. In this paper, we sustain that semantic technologies are also a promising means for achieving interoperability in co-simulation.
Co-simulation is a technique for simulating complex and heterogeneous sociotechnical systems, that is to say, systems composed of many subsystems. These subsystems may belong, not only to different parts of the physics,for example hydraulic, mechanic or electronic, but also to sociological domains, such as human behaviour.
During a co-simulation, each Simulation Unit (SU) keeps its state locally and its own simulation time. So, the key concept for performing a co-simulation is the Master Algorithm (MA).The MA coordinates SUs by exchanging data and controlling the progress of time. That means the MA advances the time of all units at the same pace (”time step”) after having transferred data between coupled units.
Complex and heterogeneous systems cannot be simulated using a single tool. Typically, it is necessary to use many tools in order to be able to export active components. In this context, the standard Functional Mock-up Interface (FMI) Blochwitz et al. (2011; 2012), Consortium et al. (2012), Association et al. (2014) appeared in 2008 to provide an interoperability framework between different simulation tools. This standard has been developed by an European consortium, and since its creation, two versions of the specification have been released, the last one being version 2.0 in June 2014. Development of the standard continues at the present time.The
FMI standard allows a system simulator to be created by co-simulating subsystems developed in different modelling tools. It defines an interface that every SU must comply with and a specification to package it in an executable file known as Functional Mock-up Unit (FMU). Likewise, the FMI standard refers to the MA as being in charge of coordinating the execution of FMUs, but is not part of the FMI standard itself.
In this sense, FMI just provides the syntactic mechanisms to allow interoperability. However, data exchange between two FMUs requires that the data be represented with the same meaning, that is to say,with the same semantic.
In this paper we discuss semantic interoperability mechanisms that may be interesting to include in the FMI. To this end, we define what semantic interoperability is understood to be and explain some of the use cases we have identified. Some of them are already supported by the FMI standard, but others are not yet fully supported. As an experiment, we have designed a Domain Specific Language (DSL), called MasterSim. This DSL is oriented to defining how SUs are connected, including support for defining semantic rules for interoperability

and taking advantage of semantic properties defined in the SUs.

Semantic Interoperability

Semantic interoperability requires data to be exchanged with an unambiguous and common meaning Heiler (1995). It is not only about packaging the data (syntax), but having data with the same meaning (semantics). While, syntactic interoperability refers to the packaging and transmission mechanisms for data, semantic interoperability refers to the meaning of exchanged data that allows it to be interpreted correctly.
To illustrate the semantic interoperability problem, there is a co-simulation example in which two SUs are connected, representing an environment and a building, respectively. In this case, we could think of a building simulation that calculates its temperature, using the environment simulation temperature (use case #1). A semantic problem could exist if the output temperature of the environment is expressed in Celsius degrees and the building expects this data to be provided in Fahrenheit degrees. In this case, there is a semantic interoperability problem.
This problem arises because the SUs are developed by different experts who are not coordinating their work. Typically, the SU developer is working in isolation and makes the design expecting that input data will be provided correctly, while not worrying about how other SUs need the output data.
In general, in an information system whose autonomous components are exchanging data, it is necessary to make the semantic of the data explicit Sciore et al. (1994). In the case presented, this problem can easily be overcome if the MA identifies that both SUs are expressing the temperature in different scales, and takes charge of converting the output coming from the environment to Fahrenheit degrees, as expected by the building. In this way, the MA would be not only responsible for exchanging, but also processing data.
This interoperability issue is addressed by FMI, which allows defining the unit of a variable. However, the standard recognises that “unit handling is a difficult topic and there seems to be no method available that is really satisfactory for all applications” Consortium et al. (2012). The unit definition in FMI consists of the exponents of the base units (kg, m, s, A, K, mol, cd and rad) and a factor/offset to allow a value with respect to unit (UV) be converted with respect to base unit (BUV) by the equation ($BUV = factor * UV + offset$)
This mechanism opens up the opportunity to perform unit checks or unit conversion. So, an input variable should have the same unit as the output variable. Furthermore, it is possible to make the conversion in the case that both variables are not represented by the same unit. The MA could perform an analysis in order to confirm that all the connections are valid, or include a simple data conversion, if necessary.
Another type of semantic analysis that is feasible with the current version of the standard is the verification of boundary conditions. FMI 2.0 allows defining the minimum/maximum value of a variable. Using these FMU properties, the MA could analyse the graph to check that boundaries of two connected variables match, and check dynamically during data exchange that the SUs are submitting or receiving values within the defined boundaries.
So, the FMI standard includes some semantic definitions that are useful for guaranteeing consistency and composability. However, it is interesting to formulate the following question: does the FMI standard provide all the necessary definitions for validating interoperability from a semantic point of view? In this paper, some use cases are analysed in order to provoke discussion of this question. Currently, the standard is being reviewed, as a new version is going to be released. In this sense, many papers that discuss features the standard lacks are considered Cremona et al. (2016), Dad et al. (2016), Tavella et al. (2016). However, we have not seen any that offer a deep study of the lack of features in the standard for performing semantic interoperability.

Semantic Requirements

Though the MA is not part of the FMI standard, it is important to consider it for eliciting interoperability requirements. In Broman et al. (2013), it is mentioned that the definition of the model structure declaring how SUs are connected to other SUs is necessary for the implementation of the MA. That is to say, the MA receives, as input, the definition of a graph which represents how SUs are coupled.
Typically, this graph could be semantically annotated to specify more precisely how to interpret the connections in the graph. These semantic annotations would allow validating data exchange between SU variables.
A typical use case is the AC modelling circuits, in which voltage, current and impedance are represented with complex numbers to express two dimensions: amplitude and phase shift. As the FMI standard does not support complex numbers, two variables are typically defined and therefore coupling between SUs must be done in pairs. So, in this case, a graph connection between two SUs should be annotated in order to interpret that two output variables should exchange data with other two input variables.
Another interesting use case we wish to discuss is related to the cardinality of connections. This is an aspect that is not fully discussed in the standard but which is very common. Typically, connections between SUs are one-to-one, though it is also possible to consider one-to-many and many-to-one connections. That is to ask, can an output be used to feed several inputs? Or, can an input be fed by several outputs? In this sense, fan-

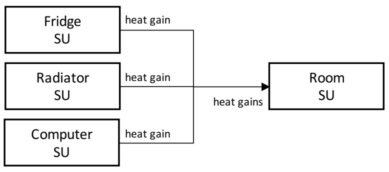

Figure 1: The Room FMU calculates the temperature based on the appliance's heat gains contained in the room

in and fan-out properties of SU variables could be used as semantic restrictions. In addition, the case of an input that is fed by many outputs requires an aggregation function to be defined. In Figure 1, an example of this use case is presented.

The last use case is the specification of a condition that should exist between two or more variables. That is to say, the value of a variable must be coherent with the value of other variables. For instance, consider a solar panel that has two input variables: tilt angle and radiation, along with energy production as output variable. In this case, it is expected that radiation is correlated with the time of day and should fluctuate slightly. This example is extensive to variables that are semantically linked but are located in different SUs or variables that should follow a pattern of variation.

These use cases are further studied in the use case section, titled as use case #2 #3 and #4 respectively.

MasterSim

In order to validate these requirements, several experiments have been conducted. Instrumentally, for describing co-simulation graphs of these experiments, we have developed MasterSim, a Domain Specific Language (DSL). This DSL allows SU connections to be defined, including the possibility of easily defining SU's semantic restrictions. Once the co-simulation graph is written using this DSL, the MA is able to validate that this graph is semantically correct,and to suggest changes in the graph to make it semantically correct, or to execute the co-simulation checking the defined semantic restrictions.

Typically, MAs are implemented in two phases: initialisation and execution. In these experiments with MasterSim, we have included an initial phase in which the graph is syntactically and semantically validated. Furthermore, during this phase,the MA is able to automatically suggest a modified graph which is valid from a semantic point of view. This is quite important when different developers are working in different aspects of a complex and heterogeneous system.

MasterSim has four main object definitions: `Cosimulation`, `SimUnit`, `Variable` and `Link`. Additionally, two facets can be applied to the variables: `Input - Output` and `DataType` (figure: 2).

`Cosimulation` can be parameterised with start time, stop time, step size and variables to be logged during the simulation. The first three parameters are oriented

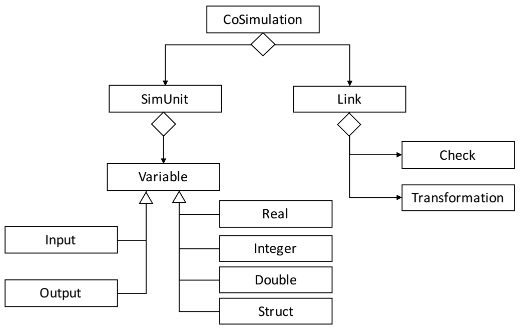

Figure 2: MasterSim metamodel

to defining the time range of the simulation and the step size. The only mandatory parameter to be given is the step size.

`SimUnit` allows defining the FMU file, from which the SU will be loaded, as well as the variables that are necessary for the data exchange. The file is given as a string, but the DSL interpreter checks whether the file ends with the expected extension (.fmu) and if the file exists.

`Variable` is an inner component of `SimUnit`. `Variable` allows definition of the variables of the SU. Inside `Variable`it is possible to define `Unit` property according to the standard by indicating attributes as exponents of base units, factor and offset. Additionally, the value range that a variable may have can be defined by using the property `Range` which is parameterised, providing the minimum and the maximum values.

These variables can be defined as `Input - Output` or `Real - Integer - Boolean - String - Struct`. `Input - Output` are oriented to defining the causality of the variable, whereas the others define the data type. When a variable is tagged as Input, it is possible to define its `FanIn` (indicating if it is single or multiple) and, when it is tagged as Output, the `FanOut` can be defined as single or multiple to indicate how many times an Output can be used. Both parameters `FanIn` and `FanOut` are statically checked by the MA so that connections with a cardinality that does not match the definition are rejected. `Struct` variables allow grouping different types of variables, so that they can be linked to other `Struct` variables that have the same composition. In this way, the MA can statically check variables that must be connected and reject the connection if such constraints are not fulfilled.

`Link` allows definition of the data exchange between variables. It has two main parameters which are called `from` and `to`. The `from` parameter requires from 1 to n variables, which must have been marked as `Output`. The `to` parameter requires from 1 to n variables which must have been marked as Input. There is an optional parameter, named `operation`, that allows definition of how the values of `from` attribute are going to be aggregated. From a semantic point of view, the `from/to` parameter cardinality must match `SimUnit FanIn` and `FanOut` properties.

```
CoSimulation(startTime = 0, stopTime = 10, stepSize = 0.01)
    SimUnit environment
        Variable temperature as Real Output
            Unit("F", K=1, factor= 0.5555, offset = 260.93)

    SimUnit building
        Variable externalTemperature as Real Input
            Unit("C", K=1, factor= 1, offset = 273.15)

    Link(environment.temperature, building.externalTemperature)
```

Figure 3: Use case #1 MA made with MasterSim DSL, option 1

```
CoSimulation(startTime = 0, stopTime = 10, stepSize = 0.01)
    SimUnit environment
        Variable temperature as Real Output

    SimUnit building
        Variable externalTemperature as Real Input

    Link(environment.temperature, building.externalTemperature)
        Transformation(@fahrenheitToCelsius)
```

Figure 4: Use case #1 MA made with MasterSim DSL, option 2

Additionally, there are two `Link` properties that can be used to improve the semantics: `Transformation` and `Check`. As its name indicates, `Transformation` property allows processing the value of the output variables of the link. This could be used as an explicit method to make transformation in data exchanges. This property is useful for dealing with the use case #1. `Check` property allows the definition of a function that is executed every time step in order to check that output and input values are semantically correct. This property is useful for dealing with the use case #4.

Use case #1

In this use case, there are two SUs: environment and building. The output temperature of the environment is given in Fahrenheit degrees whereas the input temperature expected by the building (externalTemperature variable) should be expressed in Celsius degrees. To deal with this issue, we present two different solutions. In Figure 3, the first option is presented, in which both SUs are described using MasterSim with the corresponding temperature unit, as would be done using the FMI standard.
In the second option (figure: 4), both SUs are defined, but in this case, the variables do not specify which unit they are internally expressed in. In this case, the Link is defined with the Transformation property, which includes a native code (programmed in Java) for transforming Fahrenheit degrees to Celsius. Here, the MA will execute the transformation code before setting the value in the building SU.

Use case #2

In this use case, we face the problem of the connection of variables that are, by their nature, complex numbers. As previously mentioned, in the FMI, complex numbers are normally modelled as two real variables, which requires the definition of two links, one for each part. However, from a semantic point of view, there is just a

```
CoSimulation(startTime = 0, stopTime = 10, stepSize = 0.01)
    SimUnit source
        Variable r_voltage as Real
            Range(min = -5, max = 5)
        Variable i_voltage as Real
        Variable voltage as Struct(r_voltage i_voltage) Output

    SimUnit cable
        Variable r_voltage as Real
            Range(min = -6, max = 6)
        Variable i_voltage as Real
        Variable voltage as Struct(r_voltage i_voltage) Input

    Link(source.voltage, cable.voltage)
```

Figure 5: Use case #2 MA made with MasterSim DSL

```
CoSimulation(startTime = 0, stopTime = 10, stepSize = 0.01)
    SimUnit radiator
        Variable heatGains as Real Output

    SimUnit fridge
        Variable heatGains as Real Output

    SimUnit room
        Variable appliancesHeatGains as Real Input(Multiple, @sum)

    Link(radiator.heatGains fridge.heatGains, room.appliancesHeatGains)
```

Figure 6: Use case #3 MA made with MasterSim DSL

single link and the restriction of exchanging real-to-real and imaginary-to-imaginary data.
We have addressed this issue with MasterSim in an example based on AC circuits, in which the variables have real and imaginary parts (e.g. voltage). In this circuit (figure: 5), a source provides a Struct Output called voltage, which uses the real variables r_voltage and i_voltage, referring to the real and the imaginary parts. The other component, the cable, has been defined with a Struct Input that uses the internal variables r_voltage and i_voltage. Additionally, a dependency has been defined to link the source and the cable. This way, the MA will make the data exchange of the variables contained in the structs at the same time.

Use case #3

Recalling the example in Figure 1, there are radiator, fridge and room SUs. The idea in this example is to calculate the internal temperature of the room based on the heat gains provided by the appliances that are located inside. In this use case, we are modeling it as in Figure 1 which allows "plugging" and "unplugging" devices on the fly to the input heat gains of the room.
In Figure 6, this use case is addressed using MasterSim. Three SUs have been described. The first two are appliances whose only output is the heat gains. The third one has one input called "appliancesHeatGains" that will include the heat gains of the radiator and the fridge. Note that this input variable has been defined with a Multiple fanIn and a sum based aggregation. Finally, the Link construct is instantiated to define this connection.

Use case #4

In this last use case, we address the semantic dependency that may exist between the value of different variables. In the example stated before, there was two SUs:

```
CoSimulation(startTime = 0, stopTime = 10, stepSize = 0.01)
    SimUnit radiation
        Variable value as Real Output

    SimUnit pv
        Variable radiation as Real Input

    Link(radiation.value, pv.radiation)
        Check(@checkRadiation)
```

Figure 7: Use case #4 MA made with MasterSim DSL

one providing solar radiation and another taking this solar radiation to calculate the production of a photovoltaic cell. In Figure 7, this example is addressed using MasterSim. As can be seen, there is a radiation SU that is used by the photovoltaic cell ("Pv.fmu"). This SU has the radiation as input. In the Link, a native code has been defined (checkRadiation) to check if radiation values are correct or not, in regard to what is expected. Whenever this checking reports that there is a problem, the simulation will be stopped and a message will be displayed to inform the user of the MA.

Discussion

This paper is focused on the analysis and study of use cases highlighting some requirements for semantic interoperability in co-simulation. At this moment, these requirements are not supported by the FMI standard. However, this paper is not oriented to proposing specific changes in the standard, but rather to opening a discussion on requirements that should be addressed by the standard for semantic interoperability.
These requirements (and others that could be considered in the future) are addressed in this paper using MasterSim. In this way, we have been able to express semantic restrictions to be checked either statically or dynamically. This way of expressing the semantic restrictions could be considered as a global definition, since these restrictions are not expressed in each FMU but in the MA. However, we assign the control of defining these kind of restrictions to FMU modellers. In this way, the FMI could provide mechanisms to semantically restrict the usage of an FMU.
The outcome of this paper is not only the requirements in semantic interoperability, but also MasterSim DSL. Although we have used MasterSim to address the issues presented here, MasterSim is an open source solution to create the MA for coupling SUs, and can be used by everyone, as it is available for downloading in Evora et al. (2016).

Conclusion and future work

This paper addresses the challenge of the semantic interoperability for co-simulation. It is based on the fact that each SU is independently developed and, at the development time, knowledge about how the integration should be performed might be limited, but it exists and should be considered.
We have presented a set of co-simulation use cases and requirements that describe semantic interoperability issues not supported by FMI. In addition, some experiments have been conducted to analyse these requirements. Using a DSL that we have called MasterSim,the interoperability restrictions and properties of the SU have been expressed. The MA developed includes an initial phase that is able to check all the identified semantic properties.
We propose that the FMI standard should consider these requirements. In this way, the SU could be packaged into an FMU, including restrictions and properties that could be interpreted by the MA to perform the integration. This will enable co-simulation MA to validate connections between SUs and suggest how integration should be done.
This work could be extended to consider other semantic interoperability use cases, as well as formalising the interoperability mechanisms that the FMI standard could consider in future versions. Ideally, in future versions of the FMI standard, FMUs defining semantic restrictions should be possible, in order that MAs can use this information.

REFERENCES

Association M. et al., 2014. *Functional mock-up interface for model exchange and co-simulation. Report Version*, 2.

Blochwitz T.; et al., 2012. *Functional mockup interface 2.0: The standard for tool independent exchange of simulation models.* In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany.* Linköping University Electronic Press, 076, 173–184.

Blochwitz T.; et al., 2011. *The functional mockup interface for tool independent exchange of simulation models.* In *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany.* Linköping University Electronic Press, 063, 105–114.

Broman D.; et al., 2013. *Determinate composition of FMUs for co-simulation.* In *Proceedings of the Eleventh ACM International Conference on Embedded Software.* IEEE Press, 2.

Consortium M.; et al., 2012. *Functional Mock-up Interface for Model Exchange and Co-Simulation–Version 2.0 Beta 4, August 10, 2012. Available fro m https://www fmi-standard org.*

Cremona F.; et al., 2016. *FIDE: an FMI integrated development environment.* In *Proceedings of the 31st Annual ACM Symposium on Applied Computing.* ACM, 1759–1766.

Dad C.; et al., 2016. *Parallelization, Distribution and Scaling of Multi-Simulations on Multi-Core Clusters, with DACCOSIM Environment.*

Evora J.; et al., 2016. *MasterSim DSL - information and download web site.* URL `https://bitbucket.org/siani/mastersim`.

Heiler S., 1995. *Semantic interoperability. ACM Computing Surveys (CSUR)*, 27, no. 2, 271–273.

Sciore E.; et al., 1994. *Using semantic values to facilitate interoperability among heterogeneous information systems. ACM Transactions on Database Systems (TODS)*, 19, no. 2, 254–290.

Tavella J.P.; et al., 2016. *Accurate and Fast Hybrid Multi-Simulation with the FMI-CS Standard. Accepted in Emerging Technologies and Factory Automation (EFTA) 2016 conference.*