Modelling Network Connections in FMI with an Explicit Network Model

Luis Diogo Couto[1] and Ken Pierce[2]

[1] United Technologies Research Center, Ireland
CoutoLD@utrc.utc.com
[2] Newcastle University, United Kingdom
kenneth.pierce@newcastle.ac.uk

Abstract. The Functional Mock-up Interface (FMI) is an emerging standard for co-simulation of models packaged as Functional Mock-up Units (FMUs). The FMI standard permits FMUs to share data during simulation using a simple set of types (Booleans, integers, reals and strings). FMI was primarily designed to connect continuous-time models (described as differential equations) for simulation of physical processes. The application of FMI to cyber-physical systems (CPSs) requires models of networked, discrete-event (DE) controllers. This presents a challenge for modellers as the simple FMI interface does not provide abstractions for modelling network communications. In this paper we present a discussion of the limitations of FMI in modelling networked controllers, and demonstrate a proof-of-concept of one possible solution in the form of an explicit model of the network medium called the *ether*. We describe an example ether FMU implemented in the VDM (Vienna Development Method) modelling language and its application in a smart building case study. We discuss the limitations of the approach and consider its potential application in other cases.

Keywords: Functional Mock-up Interface (FMI), co-modelling, co-simulation, network modelling, ether pattern, Vienna Development Method (VDM), Overture

1 Introduction

The design of cyber-physical systems (CPSs) requires engineers from diverse disciplines to co-operate to build and integrate a wide range of physical and software components. Model-based design is increasingly popular for building individual components, but each engineering discipline has its own paradigms and formalisms. Software engineers typically use discrete-event (DE) formalisms such as VDM (Vienna Development Method), whereas mechanical engineers use continuous-time (CT) formalisms described in differential equations. One way to allow these engineers to work together is through collaborative modelling, in which constituent models are connected together to form a *multi-model*. Such multi-models allow for analysis of CPSs at a system-level.

The Functional Mock-up Interface (FMI) is an emerging standard for co-simulation of multi-models, supported by platforms such as INTO-CPS[3] [3, 4, 6–8]. In FMI, each

[3] https://into-cps.github.io/

constituent model is packaged as an executable unit called a Functional Mock-up Unit (FMU). A multi-model comprises a set of FMUs and a configuration file describing the composition of the FMUs, their connections, and details of the simulation (e.g. duration).

FMI is an attractive proposition for industry due to its simple interface and increasing number of tools that support FMU generation[4], however FMI has certain limitations for software engineers. FMI was originally designed for co-simulation of CT models of vehicle components and because of this, it lacks abstractions to support networked, discrete-event (DE) controllers. In this paper we present a possible solution to overcome these limitations, without extending the FMI standard, by introducing an FMU that explicitly models a network medium, following a approach that we call the *ether* pattern.

In the remainder of this paper, we discuss the limitations of communications modelling in FMI in Sect. 2. We describe the ether pattern and a proof-of-concept FMI implementation using VDM in Sect. 3. We present a building case study of a Fan Coil Unit (FCU) using the ether pattern in Sect. 4. Finally we present conclusions in Sect. 5.

2 Limitations of Communication Modelling in FMI

In this section we first consider the current limitations of modelling communications with FMI. For the purposes of this paper, we consider *communication* to be messages being exchanged between two or more entities (each of which is packaged as a distinct FMU), as illustrated in Fig. 1. These messages must conform to some user-specified structure, whether it be something low-level such as packet structure or something higher level such as API calls.

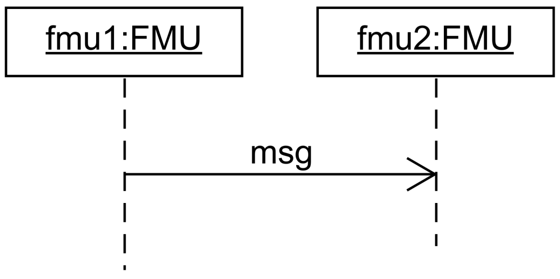


Fig. 1. Communication between two FMUs.

Before proceeding, it is worth explaining why we reason about communication *between* FMUs. The VDM-RT notation has native support for reasoning about communication using constructs like the `BUS` class and implicit remote operation calls [9, 12]. However, it is not currently possible to export only parts of a VDM model (such as classes deployed to a particular CPU) as an FMU and maintain communications. The only way to export a VDM-RT model is to export the entire model including the **`system`**

[4] A list of FMI compatible tools can be found here: `http://fmi-standard.org/tools/`

class. As such, the distribution of entities to CPUs is not visible at the FMI level. In a pure FMI setting, it is only possible to leverage the communication reasoning abilities of VDM-RT (or other modelling notations) to analyse intra-model communication. Thus, if one is interested in communication between models/FMUs, it is necessary to model the communications at the FMI level in some way.

The largest issue affecting FMI communication modelling is a fundamental one in the design of FMI— it only supports the exchange of continuous signals. This is related to the purpose of FMI, providing a means to connect simulation tools of different physical domains, all of them continuous. As such, the features and capabilities of the standard are geared towards the exchange of physical signals. The communications that we are interested in modelling are discrete phenomena: message pairs, packets, etc. However, at the multi-model level, all connections between FMUs provide continuous exchange of data. At every time in the simulation there is a value for each signal (even if that value is zero). To model concepts such as the sending and receiving of discrete-event messages, we must do so inside the models used to generate the FMU themselves. This is unintuitive as communication is primarily an inter-FMU phenomenon.

Modelling of communication can be distributed and embedded across the various FMUs, but this brings its own issues. Firstly, it will pollute the various models with communication concepts (such as message delays, response handling, etc.). Secondly, it makes it difficult to reason about communication since it is distributed across various models and hidden at the co-simulation level, where we can only observe the values of exchanged signals. We need an explicit model of the communication medium to enable reasoning. As we will see in Sect. 3, the ether pattern provides one way to do this. Another fundamental issue is that currently FMI connections are one-to-one (direct connections between two FMUs). When modelling communications, we are often interested in modelling many entities communicating between each other. This leads us to very large connection mappings between FMUs that become hard to maintain.

Assuming one has decided to model communications inside an FMU, either using the ether or distributing the communications, there are still some issues to consider, most of which have to do with the limited amount of data types supported by FMI. FMI does not support structured data types (only primitive types such as integers, reals or Booleans). However, structured data is important for modelling both low-level concepts (receiver identifier, timestamp, message data) and high-level concepts (API endpoints, payloads). Even if we model these aspects using structured data types in each constituent model, without structured data types in FMI, these aspects cannot be easily lifted to the co-simulation level.

There are two workarounds to achieve structured data types in FMI co-simulations. The first is use a collection of interdependent ports to represent each part of the data, the second is to encode/decode the structured data using strings. Both approaches have limitations. For the multi-port approach, the interdependence between the ports cannot be captured in the FMI standard, so it must be disseminated manually to the team members which leads to potential coordination issues. In addition, it also scales poorly, as complex messages will require several additional ports per FMU, which increases the complexity and effort of connecting FMUs to set up co-simulation. A high number of ports may also degrade co-simulation performance when compared to an approach using a single port.

The string-based approach scales better as everything is contained in one signal (per one-way communication channel). But this approach also has coordination issues. Namely, it is essential to share the encoding and decoding between all the models. This can be simplified by adopting a standard notation such as ASN.1 (which includes a set of Generic String Encoding Rules) as in Fabbri et al. [2]. However, the encoding and decoding functionality must still be implemented in all models. Since FMI is necessary to couple these models, they are unlikely to be able to reuse each other's functionality which will lead to duplicate implementations. Furthermore, coding and decoding messages is the kind of low-level task that we wish to avoid when modelling. This approach also requires some built-in support for strings from the modelling tools (both in terms of exporting FMU with string ports, and processing and manipulating strings within the model).

Overall, support for communication modelling in FMI is rather limited and we must make trade-off decisions between several approaches, each with their own flaws. Each approach introduces additional failure points, particularly in terms of team synchronisation. These failures can affect the co-simulations negatively.

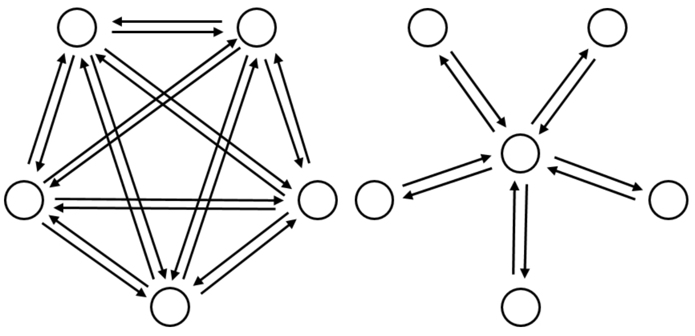
3 Modelling Networked Controllers in FMI

In this section, we address the problem of modelling networked controllers in a multi-model context. The previous section outlined the difficulties and proposed various solutions, and here we report on initial experience using the following of the solutions proposed above: using the *ether pattern* to introduce an explicit model of communication medium to the multi-model; and using string ports to marshal structured data between FMUs. In the remainder of this section we describe the ether pattern in an FMI context, report on an initial proof-of-concept ether FMU using VDM, and finally consider limitations of this FMU and identify next steps.

3.1 The Ether Pattern in FMI

When modelling and designing distributed controllers, it is necessary to model communications between controllers as well. While controller FMUs can be connected directly to each other for co-simulation, this quickly becomes unwieldy due to the number of connections increasing exponentially. For example, consider the case of five controllers depicted in Fig. 2(a). In order to connect each controller together, 20 connections are needed (i.e. for a complete bidirected graph). Even with automatic generation of multi-model configurations, this is in general not a feasible solution.

Through employing the ether pattern, a representation of an abstract communications medium is introduced (called the *ether*). This pattern was described initially in a collaborative modelling context using the Crescendo technology [5]. In an FMI setting (using the INTO-CPS technology, for example), the ether takes the form of an FMU that is connected to each controller that handles message-passing between them. This reduces the number of connections needed, particularly for large numbers of controllers such as swarms. For five controllers, only 10 connections are needed, as shown in Fig. 2(b).



(a) Topology of five controllers connected to each other (b) Topology of five controllers connected via a shared medium

Fig. 2. Connecting FMUs directly and via a shared medium

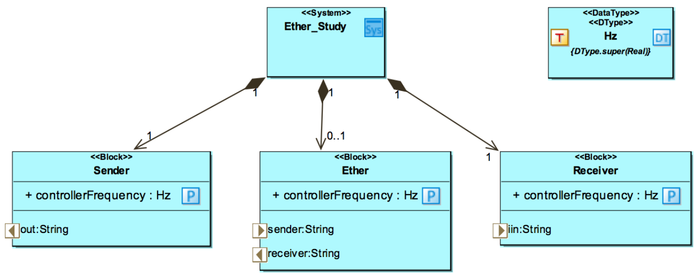
We consider a simple producer/consumer multi-model, in which a *producer* FMU sends structured data via an *ether* FMU to a *consumer* FMU. Fig. 3 shows two diagrams of this example, using views from the INTO-CPS SysML profile. Fig. 3(a) is an Architecture Structure Diagram (ASD) that shows the static structure of the system in terms of FMUs and their ports. The sender has a single output port, and the consumer has a single input port. The ether sits in between and has a corresponding port for each of these. When following the ether pattern in FMI, the ether FMU will need additional ports for new each FMU connected to it. Fig. 3(b) shows the dynamic structure of the example, with the producer connected to the consumer via the ether. This view can be used in the INTO-CPS tool chain to configure co-simulations.

3.2 A Proof-of-Concept Ether FMU using VDM

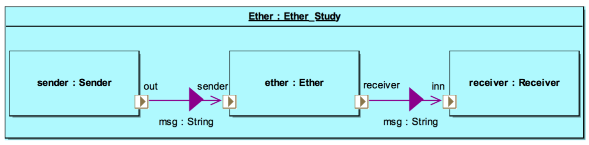
A demonstration of the ether pattern in FMI was produced by implementing the producer/consumer example above using VDM/Overture for all three FMUs (`Sender`, `Receiver` and `Ether`). The INTO-CPS technology was used to combine these FMUs into a multi-model, which is described in detail in the INTO-CPS Examples Compendium [11] and available via the INTO-CPS tool as *Case Study: Ether*.

As described in Sect. 2, connections between FMUs are typically numerical or Boolean types. This works well for modelling of discrete-time (DT) controllers and continuous-time (CT) physical models, however in a cyber-physical setting where we wish to model complex, networked controllers, it is necessary that controllers can communicate with each other using data types that are not part of the FMI specification.

In order to pass structured data between the VDM FMUs (representing the produced and consumed data), string ports were used and the data marshaled to/from their string representations within each VDM model. The Overture FMU export plug-in that enables FMU generation supports string ports, which are part of the FMI standard but not supported by all FMI-compatible tools. To encode/decone the data, the `VDMUtil` standard library was used. This can generate ASCII representation of data and parse ASCII representation back into VDM types. Using this approach it was possible to define



(a) Architecture Structure Diagram (ASD) showing static FMU structure



(b) Connection Diagram (CD) showing dynamic FMU structure

Fig. 3. SysML diagrams describing a producer/consumer multi-model that follows the ether pattern

a simple broadcast ether that receives message strings on its input channel(s) and sends them to its output channel(s). In the producer/consumer example, the three FMUs have the following roles:

Sender Generates random 3-tuple messages of type `real * real * real`, encodes them as strings using the `VDMUtil` library and puts them on its output.

Receiver Receives strings on its input port and tries to convert them to single messages of type `real * real * real` or to a sequence of messages of type `seq of (real * real * real)`.

Ether Has an input port and output port, each assigned a unique identifer, i.e. as a `map Id to StringPort`. It also has a mapping of input to output ports as a set of pairs: `set of (Id * Id)`. It has a list that holds messages for each output destination, because multiple messages might arrive for one destination. It gathers messages from each input and passes them to the outputs defined in the mapping.

In this simple example the sender and receiver are asymmetrical, but in more complicated examples controllers can be both senders and receivers by implementing both of the behaviours described above. This example served as a baseline for the industrial case study described in Sect. 4 which also follows the ether pattern.

3.3 Going Beyond the Basic Ether FMU

The proof-of-concept ether FMU presented above is basic. In each update cycle, it passes values from its input variables to their respective output variables. This essentially replicates the shared variable style of direct FMU-FMU connections. It also works as a broadcast medium, so message senders are not identified. While it demonstrates the feasibility of using the ether pattern for modelling networked controllers in FMI, there are a number of areas in which experience is needed before the ether pattern can be fully understood in this context:

Addressing This basic ether is a broadcast medium, so messages are not tagged or addressed to any specific receiver. A realistic communications medium should allow this. The basic ether does however assign an identifier to each channel however, so adding addressing is primarily a matter of adding functionality for FMUs to discover their own identifiers and those of others connected on the ether.

Timing The ether FMU in the above has a naive approach to timing, it simply passes on data that it finds on its input channels. The relative update speeds of the FMUs will affect messages, for example if a value is not read by the receiver before it is changed, then that value is lost. The introduction of an intermediate FMU means that an extra update cycle is required to pass values from sender to ether and ether to receiver. These issues don't affect the examples in this paper as the messages are high level and not time-critical, since the control loop is slower than communication speed. However if lower-level details must be simulated then a more complicated ether will be required. Useful queueing and timing principles in a co-simulation setting are covered by Verhoef [12]; in the extreme a dedicated network simulator might be required[5].

Quality of Service Realistic communications modelling will require consideration of Quality of Service. For example senders might need to wait to access a medium, or resend messages in the event of collisions. Low-level communications modelling would also require consideration of Maximum Transmission Unit (MTU), the need to break up messages, and the problems of delay, loss and repetition. By controlling for problems introduced by the nature of co-simulation, any reduction in performance of the multi-model can be attributed to the realistic behaviour introduced intentionally into the model of communications.

Reusability We do not claim that ether presented above is general purpose, however it can form the basis for work in that direction. A general purpose FMU would ideally implement the features above and be configurable for various scenarios. Libraries could be created for marshalling data to and from string representations (such as ASN.1 as suggested above).

4 Case Study: Fan Coil Unit

In this section we present an application of the ether pattern to the model-based development of an HVAC system using the INTO-CPS multi-modelling technology [10]. The complete case study covers the modeling and analysis of energy consumption and comfort levels for an HVAC system that controls the temperature of connected rooms or areas inside a building as shown in Fig. 4. It contains models of several concepts including physical rooms and air flow; Fan Coil Units (FCUs); supervisory control of

[5] Example network simulators include OpNet `www.riverbed.com/` and the open-source NS2 `www.isi.edu/nsnam/ns/`.

FCUs; communications between master-slave FCUs; communications between FCUs and a supervisor; air-pipe connections between FCUs and Air-Handling Units (AHUs); and water pipe connections between FCUs and Heat Pump Units (HPUs). For this paper we focus on the communication between FCU controllers and a supervisor.

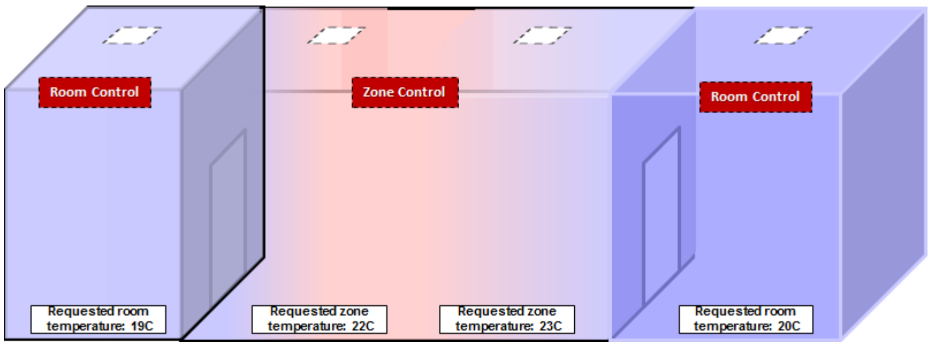

Room Control
Zone Control
Room Control
Requested room temperature: 19C
Requested zone temperature: 22C
Requested zone temperature: 23C
Requested room temperature: 20C

Fig. 4. Rooms and Zone level schematic for the case study

The relevant part of the control strategy for the system is regulation of FCU operations and supervision of FCUs. User input is taken into account from room or zone thermostats and is compared with current room air temperature sensed by the FCUs, triggering the FCUs to take a series of actions to reach the desired temperature by regulating the air flow (using its fan), and regulating the water pipe valves to control the cooled or heated air. The supervisor oversees and coordinates the behaviour of the FCUs and may override local FCU decisions.

In terms of the ether pattern, there are two relevant entities to be modelled:

– FCU controllers modelled in MATLAB Simulink
– Supervisor modelled in VDM-RT

We briefly outline the functionality of both entities. The primary task of the FCU controller is to run a PI loop to enable the FCU to control the room temperature. The FCU controllers also support a master-slave behaviour. Broadly speaking, the master overrides the set point of its slaves to ensure more consistent control over a zone. An FCU must be aware of its role as master or slave and work from the correct set point accordingly. The role of the supervisor is to assign an FCU as master (and possibly reassign it, if faults are detected), to distribute set points from masters to slaves, and to ensure that any set point in the FCUs is within certain maximum and minimum bounds.

To enable the supervisor activity, the FCU controllers expose an API for the supervisor so that it may control them using a higher level set of instructions. Namely, FCU controllers must provide getters for the set point and setters for role (master or slave) and the set point of the master.

The first iteration of the case study had a single controller. In the second iteration of the case study, multiple controllers were introduced and these were modelled entirely within a single VDM-RT model. The supervisor and the FCU controllers (including the PI loops) were modelled as separate classes and deployed to different `CPU` objects in the model. This enabled the use of the native communication facilities of VDM-RT. In the third iteration of the case study, the FCU controller model was replaced with a Simulink model, due to performance of the PI controllers. The supervisor remained in the VDM model. As such, it was necessary to connect four Simulink FMUs and a VDM model.

We adopted the ether pattern (basing our implementation on the example described in the previous section), centralising all communication in the VDM model, but adapting the pattern to also include supervisory behaviour within the ether model. Our motivation for embedding the supervisor in the ether is to reduce the amount of FMI communications. In addition, we are not particularly interested in reasoning about communication details, since their impact on temperature variation in this use case is minimal [1]. In spite of this, we found the ether to be useful as a way of structuring our multi-model and centralising communication between the Simulink and VDM FMUs.

In our case study, we implement the supervision API as a protocol between the supervisor and FCU controller. This protocol is represented by three FMI signals, as shown in Fig. 5. Originally, the entire protocol was encoded in a single port using String FMI signals. However, the FMI Toolbox for MATLAB/Simulink does not support exporting FMUs with String ports, so the values were separated as follows:

- *Mode* (supervisor to FCU): An integer that indicates the operating mode of the FCU (0 for OFF, 1 for master, 2 for slave)
- *SuperGet* (FCU to supervisor): A string that encodes the measured temperature and set point of the FCU. Format: `mk_(TEMP,SETPOINT)`.
- *SuperSet* (supervisor to FCU): A real that overrides the FCU set point.

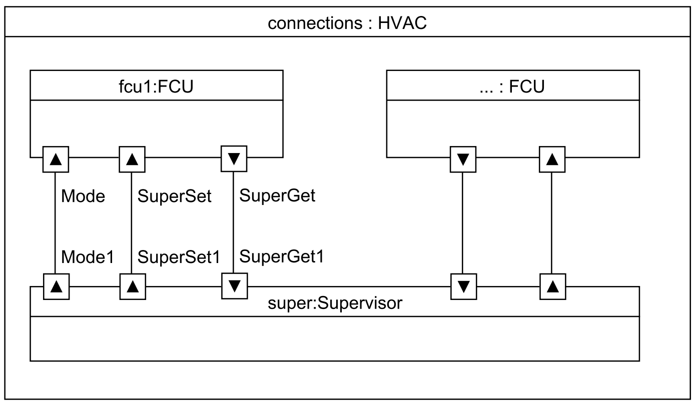


Fig. 5. FMUs communicating over FMI with the Supervisor embedded in the ether (all FCUs have the same connections).

In the Supervisor model, each FMI port is represented in the special FMI class called `HardwareInterface`, as defined in the Overture FMI export tool. Each port is then connected onto a variable of an `FCU`, which is deployed onto a separate CPU, as shown in the instantiation of the `System` class below:

```
system System

instance variables
public static super : [Supervisor] := nil;
public static sr1 : [FCU] := nil;
public static sr2 : [FCU] := nil;
public static z1 : [FCU] := nil;
public static z2 : [FCU] := nil;

public static hwi: HardwareInterface :=
  new HardwareInterface();

cpu1 : CPU := new CPU(<FP>, 5E3);
cpu2 : CPU := new CPU(<FP>, 5E3);
cpu3 : CPU := new CPU(<FP>, 5E3);
cpu4 : CPU := new CPU(<FP>, 5E3);
cpu5 : CPU := new CPU(<FP>, 5E3);
bus : BUS := new BUS (<FCFS>, 960,
  {cpu1, cpu2,cpu3,cpu4,cpu5});

operations
public System : () ==> System
System () ==
(
  sr1 := new FCU(1);
  sr1.primeFmi(hwi.sr1_spIn,hwi.sr1_spOut,hwi.sr1_mode);
  cpu1.deploy(sr1,"FCU_SR1");

  sr2 := new FCU(2);
  sr2.primeFmi(hwi.sr2_spIn,hwi.sr2_spOut,hwi.sr2_mode);
  cpu2.deploy(sr2,"FCU_SR2");
  -- ...
  super := new Supervisor({sr1,sr2,z1,z2});
  cpu5.deploy(super,"Supervisor");
);
end System
```

The `FCU` class is a very thin wrapper around the FMI ports which provides the supervisor with read and write access to the FMI values and hides all FMI aspects and side-effects (such as updating the FMI output port when a new set point is set). The `Supervisor` runs a continuous loop where it controls and distributes the set points of the FCUs and also performs fault detection, as shown in the abbreviated listing below.

```
class Supervisor
public control :()==>()
control()== (
  for all fcu in set {fcu | fcu in set dom fcus & fcu.isOn}
  do (
    if not fcus(fcu).faulty then checkDrift(fcu);
    distributeTemps(fcu);
  )
);

private distributeTemps : FCU ==> ()
distributeTemps(fcu) == (
  if fcu.role=<MASTER> then (
    let
      masterTemp = fcu.getMeasuredTemp(),
      nsp = getAdjustedSetPoint(fcu)
    in (
      fcu.setSetPoint(nsp);
      for all s in set fcu.getSlaves() do (
        s.setSetPoint(nsp); -- new value flows
                            -- to FMI variable as
      )                     -- side-effect
    )
  );
);
-- ...
end Supervisor
```

5 Conclusions

We have presented an approach to modelling networked control of CPSs in FMI. The approach consists of an explicit model of the network medium called the ether. We described the ether pattern and presented a basic, proof-of-concept implementation using a VDM model. We demonstrated its usage through a case study from the smart building domain, specifically the HVAC system.

The ether pattern is an effective way of modelling networked controllers within the context of FMI, without requiring changes or extensions to the standard. However, as we have seen from our case study, limitations in the FMI tools can prevent us from deploying it fully. In a setting where we are using VDM/Overture for all controllers, this is not a concern since Overture supports FMI strings. However, when using more heterogeneous tool chains, this may be a concern.

As our case study shows, the ether pattern can be used not only to model communication, but also to structure the FMUs of the co-simulation, particularly in terms of how it affects VDM modelling. By following the ether pattern when developing the multi-model, we can also easily extend it in the future with a more detailed model of the communication medium including concepts such as addressing, reliability and speed.

On the other hand, the usage of the ether does bring non-trivial burdens to the modeller and pollutes models with low-level communication concepts such as encoding and decoding of messages that might not be of interest. In addition, it requires support for strings, which is not certain for all FMI tools. While we have shown that the ether pattern can overcome some limitations in the modelling of networked controllers in FMI, further work is needed to investigate features for realistic communications modelling: sender identification, confirmation of delivery, maximum transmission unit, message timing and so on. In addition, the question of how multiple communication channels can exist together in a multi-model should be considered, i.e. should they be modelled within one ether FMU or should multiple ether FMUs be introduced. It is possible that some of these aspects can only be addressed directly at the level of the FMI itself through modifications to the standard, however a richer standard "ether FMU" and libraries could go a long way to helping multi-modellers work with networked controllers.

Acknowledgements

We wish to thank Pasquale Antonante and Hajer Saada for their assistance in modelling the case study. This work is supported by the INTO-CPS H2020 project: *Integrated Tool Chain for Model-based Design of Cyber-Physical Systems*. Funded by the European Commission-H2020, Project Number:664047.

References

1. Couto, L.D., Basagianis, S., Mady, A.E.D., Ridouane, E.H., Larsen, P.G., Hasanagic, M.: Injecting Formal Verification in FMI-based Co-Simulation of Cyber-Physical Systems. In: Submitted for the 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems (CoSim-CPS). Trento, Italy (September 2017)
2. Fabbri, T., Verhoef, M., Bandur, V., Perrotin, M., Tsiodras, T., Larsen, P.G.: Towards integration of Overture into TASTE. In: Larsen, P.G., Plat, N., Battle, N. (eds.) The 14th Overture Workshop: Towards Analytical Tool Chains. pp. 94–107. Aarhus University, Department of Engineering, Cyprus, Greece (November 2016), ECE-TR-28
3. Fitzgerald, J., Gamble, C., Larsen, P.G., Pierce, K., Woodcock, J.: Cyber-Physical Systems design: Formal Foundations, Methods and Integrated Tool Chains. In: FormaliSE: FME Workshop on Formal Methods in Software Engineering. ICSE 2015, Florence, Italy (May 2015)
4. Fitzgerald, J., Gamble, C., Payne, R., Larsen, P.G., Basagiannis, S., Mady, A.E.D.: Collaborative Model-based Systems Engineering for Cyber-Physical Systems – a Case Study in Building Automation. In: Proc. INCOSE Intl. Symp. on Systems Engineering. Edinburgh, Scotland (July 2016)
5. Fitzgerald, J., Larsen, P.G., Verhoef, M. (eds.): Collaborative Design for Embedded Systems – Co-modelling and Co-simulation. Springer (2014), `http://link.springer.com/book/10.1007/978-3-642-54118-6`
6. Larsen, P.G., Fitzgerald, J., Woodcock, J., Fritzson, P., Brauer, J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., Sadovykh, A.: Integrated Tool Chain for Model-based Design of Cyber-Physical Systems: The INTO-CPS Project. In: CPS Data Workshop. Vienna, Austria (April 2016)

7. Larsen, P.G., Fitzgerald, J., Woodcock, J., Lecomte, T.: Trustworthy Cyber-Physical Systems Engineering, chap. Chapter 8: Collaborative Modelling and Simulation for Cyber-Physical Systems. Chapman and Hall/CRC (September 2016), iSBN 9781498742450
8. Larsen, P.G., Fitzgerald, J., Woodcock, J., Nilsson, R., Gamble, C., Foster, S.: Towards Semantically Integrated Models and Tools for Cyber-Physical Systems Design, pp. 171–186. Springer International Publishing, Cham (2016), `http://dx.doi.org/10.1007/978-3-319-47169-3_13`
9. Larsen, P.G., Lausdahl, K., Battle, N., Fitzgerald, J., Wolff, S., Sahara, S., Verhoef, M., Tran-Jørgensen, P.W.V., Oda, T.: The VDM-10 Language Manual. Tech. Rep. TR-2010-06, The Overture Open Source Initiative (April 2010)
10. Ouy, J., Lecomte, T., Christiansen, M.P., Henriksen, A.V., Green, O., Hallerstede, S., Larsen, P.G., ger, C.J., Basagiannis, S., Couto, L.D., din Mady, A.E., Ridouanne, H., Poy, H.M., Alcala, J.V., König, C., Balcu, N.: Case Studies 2, Public Version. Tech. rep., INTO-CPS Public Deliverable, D1.2a (December 2016)
11. Payne, R., Gamble, C., Pierce, K., Fitzgerald, J., Foster, S., Thule, C., Nilsson, R.: Examples Compendium 2. Tech. rep., INTO-CPS Deliverable, D3.5 (December 2016)
12. Verhoef, M.: Modeling and Validating Distributed Embedded Real-Time Control Systems. Ph.D. thesis, Radboud University Nijmegen (2009)