Enable Co-Simulation for Industrial Automation by an FMU Exporter for IEC 61499 Models

Jose Cabral, Monika Wenger, Alois Zoitl
fortiss GmbH
Forschungsinstitut des Freistaats Bayern für softwareintensive Systeme und Services
München, Deutschland
Email: {cabral, wenger, zoitl}@fortiss.org

***Abstract*—Different causes have contributed to the shift of paradigm in the automation field from centralized to modular and distributable. Software and hardware of Programmable Logic Controllers (PLCs) have evolved to be synchronized with the demands nowadays and new standards are trying to follow this trend. One of them is the IEC 61499 standard, which is intended for modelling distributed industrial solutions with a vendor-independent format. This shifting is also bringing new challenges and this paper focuses on the virtual commissioning of plants, which can become a harder task when the system is distributed and involves different interconnected modules. A major problem when doing a virtual commissioning is the coupling between the different physical systems and controlling tools since many vendors still offer closed integrated solutions. Functional Mockup Interface (FMI) is a standard for a co-simulation interface which intends to fill the gaps between different modelling tools, by packaging the models into Functional Mockup Unit (FMU) and allowing a co-simulation master algorithm to have access to the inside model through the proposed interface. This paper presents a mapping between IEC 61499 models and the FMI standard and an implementation of a tool that can export IEC 61499 models into FMUs, which would allow the co-simulation of physical plants and the PLCs software that controls it. An experiment is shown where a simple Proportional-Integral-Derivative (PID) controller of a tank system is exported as an FMU and co-simulated together with a tank system modelled using OpenModelica.**

I. Introduction

The fourth revolution in the industry (Industry 4.0 or Smart Factory) wants to transform industry the same way the Internet transformed the computers, and even go further. Although the concept of Industry 4.0 may not be the same for all involved parts, some concepts have a strong presence. The Internet is again one of the main actors in this revolution, bringing connectivity to the smallest components in the automation chain. As the revolution is encouraging the flexibility of its parts, centralized systems are evolving to more distributed and modular ones [1].

The heart of these systems is the PLC controlling the logic of the automation. Transforming automation systems from centralized to more modular ones, triggered the creation of the IEC 61499 [2] standard, which complements the well established IEC 61131 [3] and focuses on the missing qualities of distributed systems, not present in the IEC 61131 or even needed in the times it was written.

As in other fields, the automation software to be deployed into the real systems must be tested beforehand, because stops in the production of any kind, desired or not, cost money. In the automation field, the need for these steps can be higher since an error can not only stop the production but also damage machines and even humans involved.

When one talks about virtual commissioning in the automation field, the logic of PLCs is tested against the virtual representations of the systems they are controlling. This reduces the actual commissioning time and in case of software updates, these can be safely tested. In many cases, the testing starts with Software-in-the-loop (SIL) where the software to be tested runs on a computer different to the one where it will finally run. Digital inputs and outputs are simulated and connected to a virtual system. On a second step, in Hardware-in-the-loop (HIL) tests, the software runs on the real hardware where it will run once the installation is completed. The inputs and outputs of the hardware are actually triggered, and some type of coupling is added to connect these to the virtual system to be controlled [4].

In both cases, normally both logic and the virtual representation of a system are tightly coupled, and combining tools from different vendors can be a hard task. With the new paradigm of distributed systems, the integration of different tools from different vendors is a greater need. The term co-simulation can be seen as SIL where models from different fields are simulated together by a master that controls them. One of the standards in co-simulation that has gained attention in the last years is FMI [5], which defines an interface between models that are packaged in so-called FMUs.

FMI can be very useful to automation systems, where hardware's manufacturers can offer their FMU (intellectual property can be maintained) to software developers to test their controlling software against it. FMI allows the co-simulation to be tool-independent, making the integration of several models, normal in distributed systems, an easier task.

This paper presents a mapping between IEC 61499 models to the FMI standard and an FMU exporter from a model in IEC 61499, allowing the developer to test the logic of the system against physical models from domain-specific tools.

In Section II an overview of the work being done that is related to this paper is presented, together with a brief presentation of the key players to this paper. In Section III the

mapping between FMI and IEC 61499 model is explained, to see the relation between types and other characteristics. The actual exporter is introduced in Section IV, together with the tools used for it. Section V presents an experiment that was done for testing the exporter, where a controller modelled in IEC 61499 is tested against a system of a tank modelled in OpenModelica, using a master algorithm which co-simulates both FMUs. Finally, a conclusion is presented in Section VI, where an overview of the work done in the paper is presented and possible future work that can spaw from this.

II. State of the art

In this section, the two key players of this work are presented. Besides that, related work about co-simulation with FMI is described.

A. Background on IEC 61131 and IEC 61499

During decades, the industry got used to the well established centralized architecture of automation, where one PLC controls one system. The blooming of technology in this field brought with it a number of vendors offering different solutions which made the different stakeholders look for a solution to all the interoperability problems that come with the lack of a standard. In the early nineties, the standard IEC 61131 was the first step to overcome this issue, especially in its third part, stating different programming languages for PLCs, among them, and interesting for this paper, the Function Block Diagram (FBD), which is a graphical programming language with Function Blocks (FBs) representing functions and input and output variables connected between them with lines [3].

Even though the aforementioned standard helped to fill some interoperability gaps on a high level, especially for the PLC programmer, it did not prevent some other issues. As an example, a software developed for a brand of PLC using one of the standard programming languages was not portable, or at least easily portable, to a PLC from another vendor. This means that industries were normally tied to a vendor, and the incorporation of new vendors wasn't easy since it would require re-programming of software that was already tested and implemented. Also, since the IEC 61131 standard was written in a time when the industry and the automation were designed using a centralized architecture, communication between PLCs wasn't properly covered, producing more issues when using PLCs from different vendors. This last issue was exacerbated in the following decades with the tendency of the production and automation to being more flexible.

The IEC 61499 standard [2] (from now on, when the word "standard" is alone, it will refer to IEC 61499) is a domain-specific modelling language intended for distributed systems in the automation field. It was created not as a replacement of IEC 61131, but as an extension of it. It even uses the same data types as IEC 61131. The standard describes several models which help to describe distributed automation systems on a high level. The modelling is platform independent and a eXtensible Markup Language (XML) schema is provided to overcome portability issues. Four of the models described in the standard are presented in this paper since they are of bigger interest for the objectives of it.

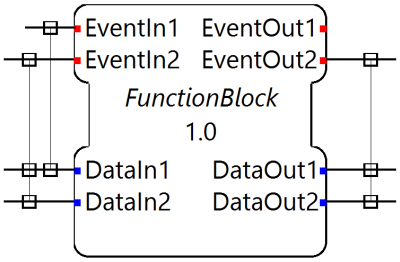

Fig. 1: Example of an interface of a Function Block

The **FB model** is the base model of the standard and it is used to represent an encapsulated functionality. In IEC 61499 global variables are forbidden. A big difference with the FB from IEC 61131 is the introduction of events, which are separated from data and they trigger the FB and control when data is refreshed, governing the sequence to be executed in a network of FB. The interface of a FB is composed of the input and output events (top of the FB), and the input and output data (bottom of the FB) and their types. This interface should be made public by the developer of the FB for their portability, but the internal behaviour can be maintained private. An example of an interface is shown in Fig. 1. There are three types of FBs and this classification is important when implementing an FMU:

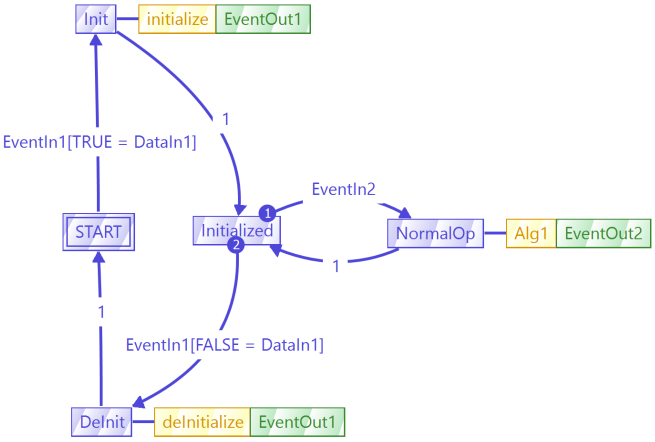

Fig. 2: Example of an ECC of a Basic Function Block

- **Basic Function Block (BFB):** their functionality is defined by an Event Execution Control (ECC) (Fig. 2), a state diagram with internal variables and algorithms written in programming languages stated in IEC 61131-3 or compliant to IEC 61499. The algorithm execution, state changes and output events are controlled by the arriving events to the FB.
- **Composite Function Block (CFB):** the internal functionality is defined by a network of FBs of any type (Fig. 3).

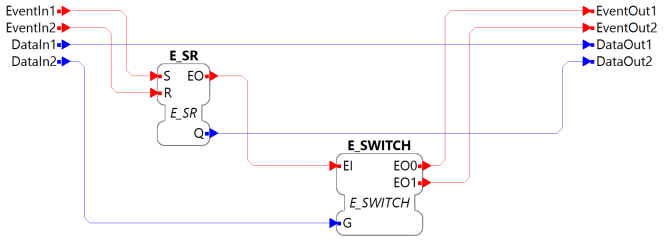

Fig. 3: Example of an internal network of a Composite Function Block

- **Service Interface Function Block (SIFB):** their behaviour is user defined and is normally used to access hardware resources or platform dependent features like inputs, outputs or communication.

The **Application model** is a network of FBs with event and data connections between them (Fig. 4). The model also specifies where each FB is executed since applications can be deployed on different devices. A chain of events is generated from a SIFB that leads the execution of the FBs, by following the output event of a FB to the input event of the next one and so on. The **Device model** is used to show the resources inside it. Applications are deployed into the resources. Resources independently (from other resources on the same device) control the execution of any FB network that has been deployed to it. This means that resources deliver any external event to the FB network they maintain, and they take care that only one event per time is delivered and no external event gets lost. Finally, the **System model** represents the devices and the relation between them in a distributed system.

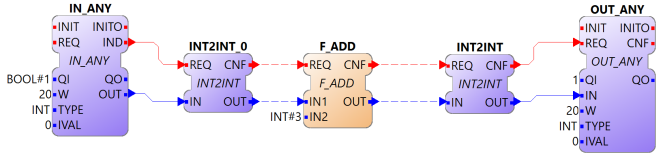

Fig. 4: Example of a simple application

B. Key aspects of FMI

The FMI [5] was developed after the need for a better interaction between models defined in different modelling tools by the partners of the MODELISAR project[1]. Nowadays, more than a hundred tools have implemented some type of FMI support[2]. The standard is in its second version, and this work focuses on this one rather than the previous one.

The interface is a set of C functions that a model must implement. The exported file is called FMU, a zip file with the ".fmu" extension. The standard details the structure of the FMU, as well as the minimum content of it. The core parts of it are a Model Description in terms of an XML file, with the definitions of all exposed variables, inputs and outputs among them, other static information of the FMU, and the binary or C files for the model that actually implements the C interface.

The FMI standard specifies the variable types to be used in the FMUs. They are defined in a header file which can be uniquely identified using the interface. The main data types are *fmi2Real*, *fmi2Integer*, *fmi2Boolean* and *fmi2String*, which are used to represent floating point numbers, signed integer numbers, boolean numbers and character strings ('\0' terminated, UTF-8 encoded) respectively. Different other types of structures and pointers are defined but are not presented here, since they are related to the internal implementation of the FMU exporter, and not to the IEC 61499 model itself. The size of the data types is not fixed but it is provided by the environment where the FMU shall be used. The variables of an FMU's model should be mapped to one of the four types presented. The C interface has functions to get and set the inputs, outputs and internal variables according to the type, e.g. *fmi2GetReal*, *fmi2SetReal*, *fmi2GetBoolean* and so on. The master algorithm knows how to reference each variable in the model by reading the Model Description XML file.

The above-mentioned characteristics are common for all FMUs, but the standard is divided into two main parts, with functions that are common for both parts and specific ones that only need to be implemented for one of these parts.

- **Model Exchange (ME):** In ME, the model is represented using equations. The FMI for ME provides the access to these equations. The tool that uses FMI for ME should then have a solver for the equations.
- **Co-Simulation (CS):** The CS has the solver in the FMU itself and could be seen as a black-box that is simulated independently. Exchange of data between FMUs for CS is restricted to discrete communication points, and are not direct, but through the master algorithm, which is in charge of controlling each independent simulation and doing the transferring of the data. The master algorithm is not part of the FMI standard. The functions for CS in the C interface allow to do a step in the simulation and cancel it. The interface also provides the means to allow the FMU to interpolate the continuous real inputs between communication points, and to retrieve the actual status of the simulation.

C. Co-Simulation with Functional Mockup Units

Co-simulation is not a new concept, and it has been a subject of studies for many years. A very comprehensive work was done in [6] where cases and studies from different disciplines are analyzed. It presents approaches, challenges and a detailed taxonomy according to different aspects of state of the art of co-simulation. A focus is made on the importance of FMI, and the difficulties that arise when the co-simulation is taken to the field of Cyber-Physical System (CPS), a key player in automation.

In [7], the authors present a co-simulation case of a CPS containing a diesel engine and its controller. The controller

[1]https://itea3.org/project/modelisar.html
[2]http://fmi-standard.org/

runs on an embedded system, so in order to simulate this, they pack the entire Real-Time Operating System (RTOS) as a library in the FMU and hook the Operating System (OS) clock to control it from the FMI interface. A similar approach is taken in [8], where the authors use TinyOS as OS. With intermediate steps which transform applications to SysML and adding an emulator of the OS, an FMU is generated. The interface of the FMU is taken from an XML file exported by the compiler of the source files. Another co-simulation test is done in [9] using Vienna Development Method - Real Time (VDM-RT), a formal method used in the development of computer systems, for the modelling of the controller of a Heating, ventilation, and air conditioning (HVAC) system.

A work using FMI and IEC 61499 is done in [10], but in this work, a model in IEC 61499 is used to handle the communication to ME FMUs, and no CS FMUs is used. In [11] the authors modelled both, a system of a box filler and its controller, using IEC 61499 and then deployed them in connected PLCs for an HIL simulation. In this case, the model of the plant is too simplified and can lack needed physical details.

Most studies which use FMI come from the physical field and the FMU of the controller is generated specifically for the study. By exporting an FMU from an IEC 61499 model, the controller is modelled in a higher level language and remains vendor-independent.

III. Mapping of IEC 61499 models into FMI for co-simulation

This section presents a co-relation between the key concepts of both standards, like inputs and outputs, variables, types and others, that will later serve as a base for the exporter. This was done by analyzing the requirements of both standards and finding the correlation between them.

A. Devices as FMUs

Each device in the System Model in IEC 61499 (normally a PLC in the system) can be mapped to an FMU, and the input and outputs of the device can be mapped to inputs and outputs of the FMU. Creating several instances of the FMU would mean creating instances of the devices with all resources and applications specific to each instance. Resources in the device are executing in parallel the chain of events of the application deployed in them. For the co-simulation FMU, the resources should be executed only when the FMI function `doStep` is called, and pausing the execution when `doStep` is not being called.

This is the big difference between FMI and IEC 61499 since FMI is time-based and IEC 61499 is event based. Also, the execution time of each FB depends on the machine where it is running, so the number of events to be executed in a chain of events during the `doStep` call can vary from machine to machine and losing determinism. In other modelling languages the concept of ticks is used to define the amount of them needed for certain code to execute. The amount of ticks per second is then defined. One possible solution could be to set the ticks per second of the device as a parameter, but then, the relation between the model and the runtime would be too tight, losing interoperability. A better solution could be to follow annexe G of IEC 61499, which defines that every element can have attributes. The ticks per second of the device where the model will be running can then be defined as an attribute of it. Another approach to a possible solution is to run the FMU in the machine where the model is then deployed. The FMI standard presents an infrastructure to distribute the co-simulation using wrappers. This implementation is out of the scope of this work, but it presents a nice opportunity for future investigations.

B. Mapping of data types

Regarding data types, the mapping must be done both ways, from IEC 61499 data types to FMI and the other way around.

The available data types in IEC 61499 can be mapped to the four data types available in the FMI. This mapping can be divided into three levels. The first one is for direct mappings, for example boolean, real and string which have a direct translation in data types in FMI. In a second level, all the variations of similar data types in IEC 61499 can be found, for example, `SINT`, `DINT`, `USINT` and so on, which represent values for different combinations of the signed and size characteristic. These are mapped to the corresponding data types in FMI similar to the first level. And on the third level, all the time-related data types are found. Since the store is vendor dependent, but the representation is given in IEC 61131-3 they are directly mapped to strings in the FMI.

The other way around, from FMU to IEC 61499 data types, the best approach is to use the biggest container for each case, since the actual type of the FMU depends on the exporter. So for example, an integer from the FMI can be mapped to `LINT`, a long integer of 64bits. But this representation must be consistent only inside the FMU, so it can vary among exporters.

C. IEC 61499 information available in the FMU as variables

The variables that the IEC 61499 model will make available as an FMU will depend on the type of FB being used. For all cases, the interface is always public, meaning that all data inputs and outputs can be offered as variables. Events cannot be directly mapped to variables, because of their nature and they don't have a data type, but the number of times an event has been triggered can be treated as an integer in the FMU. Since the IEC 61499 standard doesn't allow two instances of FBs to have the same name, the name of the variable in FMU could start with the name of the FB, followed by a concatenating character (for example a point) and the name of the data point or event (which is also unique in the FB). From the example in Fig. 4, the data inputs of the `F_ADD` FB will be called `F_ADD.IN1` and `F_ADD.IN2` and will be of type integer. For FBs which are distributed by the developer as a black box, no other information is available.

For BFB internal variables and the state of the ECC can be offered in the FMU. The ECC can be a string with the

name of the state, or an integer with the index of the state according to the representation in the IEC 61499 XML. Since internal variables cannot have a name that is already used in the interface of the FB, the concatenation approach still holds. For the ECC, a unique name could be chosen, which is not present in the interface.

For CFB all the internal FBs according to the case can be variables. The method of using concatenation for the name of the variable also applies to this case.

SIFBs are a special case since their implementation is in most cases hardware specific. On one side, they are used to find the Input and Outputs (IOs) of the model. FBs like `IX`, `QX`, `IW` and `QW`, defined in the standard, should be treated as the input and output of the FMU. As an example, the Boolean value coming from the FMU should be present in the `IN` data type of the `IX` FB, as it is illustrated in Fig. 5. The `IND` event should be triggered when there's a new value in the input. The method for detecting the change in the input, either polling or triggered by hardware, should be the same as in the runtime environment that the actual machine will be using. Similar to IO FBs, communication FBs could be treated as additional Inputs and Outputs, but this brings new issues especially on identical incoming packets. Also, communication is a very important feature in distributed systems, so its full analysis is out of the scope of this work. In this work, communication FBs are omitted from the examples. For all other SIFBs, they should be avoided or have an internal twin representation in the runtime environment that is running the FMU.

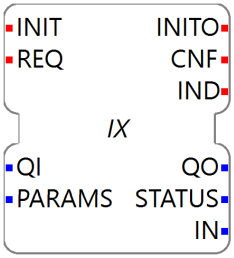

Fig. 5: IX Function Block

IV. Eclipse 4diac framework as platform example

The Eclipse 4diac framework[3] is an open-source infrastructure for developing and executing IEC 61499 models. It was used for developing the FMU exporter, so its main parts are presented:

- 4diac-ide: it is an Eclipse-based Integrated Development Environment (IDE) written in Java, which allows the user to design distributed automation systems according to IEC 61499 in a graphical way. The user can create applications, devices, resources, map them and also create their own FBs.
- tool library: the library contains commonly used FBs, like the ones that are defined in the standard itself, FBs that are equivalents from the ones IEC 61131-3 and some others proven from experience to be useful.
- 4diac-rte: also known as forte, this runtime environment written in C++ creates devices, loads resources and application definitions, and executes them according to the IEC 61499 standard. It was ported to many platforms, among them to several OSs of PLCs.

[3]https://www.eclipse.org/4diac/

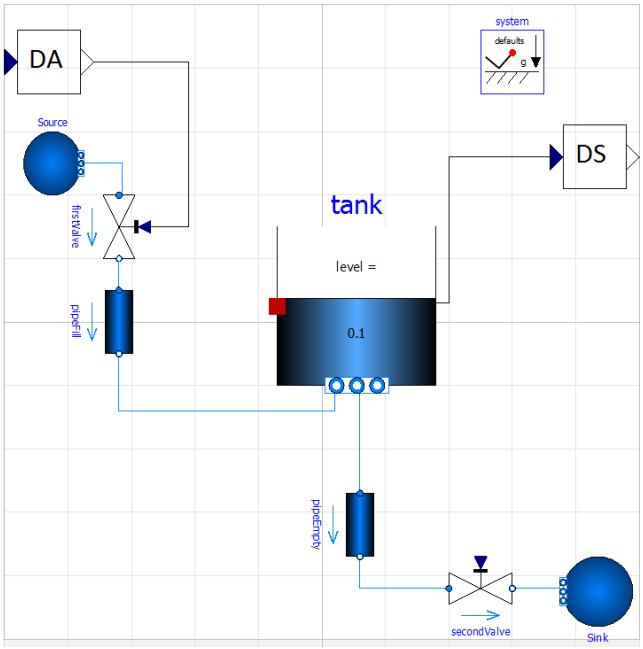

Fig. 6: Tank model in openModelica

To create the FMU, the approach for the exporter was to create the Model Description for it and a definition of the IEC 61499 model from 4diac-ide and use an adapted 4diac-rte for the binaries files.

A. FMU Exporter on 4diac-ide

The IDE was extended with a plugin for exporting one FMU for each device of a system. It creates an FMU for each device doing the following tasks:

- Analyze the FBs and resources of the device and creates the Model Description following the criteria specified in section III.
- Export the model of the device in the IEC 61499 XML format, a feature that was already implemented.
- Creates the FMU by creating the folders according to the FMI standard, locating the IEC 61499 model in the resource folder, and the binaries (see section IV-B) in their respective folders and zipping them all together.

The plugin allows to export several devices in different FMUs at the same time and allows the selection of binaries from different platforms (Windows, Linux, x86, x64) to be included on the FMUs.

B. FMU Exporter on 4diac-rte

The FMI feature in 4diac-rte was realized by:

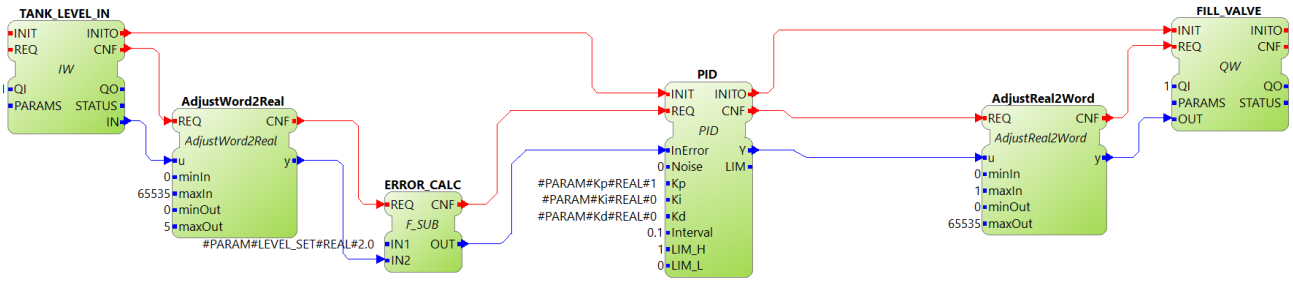

Fig. 7: Controller model in 4diac-ide

- Adding a process interface for handling the Inputs and Outputs, which basically stores the value in an internal variable instead of accessing the hardware interface in real PLCs.
- Adding a communication layer (optional) which mimics the underlying communication interface by working in a similar way as the added Inputs and Outputs process interface.
- Adding special FBs for parameters of the FMU since these should be set before starting the application.
- Enabling a feature that allows the resources to be paused and resumed.
- Adding the FMI functions that execute the procedure according to the FMI standard.

One big change that was required to implement the exporter on 4diac-rte was to get rid of all present global variables in the code. For example, the current time was one of them, but since the FMU uses a library and many devices can be instantiated from it, global variables will be shared between them, creating problems in the co-simulation.

V. Experiment and Results

To test the exporter in a co-simulation environment, a simple system of a water tank and its controller were modelled. The water tank system, as illustrated in Fig. 6, has a valve that is always opened that the fluid is able to escape from the tank. To fill the tank, another valve is provided, whose opening can be controlled and the high limit of the opening is bigger than the constant opening of the other valve, allowing the control system to fill the tank to the desired level. The opening of the filling valve together with the current level of the tank are provided as input and output of the tank system respectively, although they are not directly provided, but through dummy sensors and dummy actuators that convert the physical measurements from and into 16Bits unsigned values, which are normally used as analog values in PLCs.

The water tank was modelled using openModelica[4] and the controller in 4diac-ide. The aim of this example is to test the co-simulation between different tools, one of them being the exported IEC 61499 model, so the level of details of the model is kept low.

The controller of the tank was modelled in 4diac-ide, whose core was a BFB which implements a PID control. One FB for each analog input and analog output were used to connect to the tank system. The desired tank level and the constants of the PID controller `Kp`, `Ki` and `Kd` were set as parameters in the FMU in order to do a proper tuning of the controller. The application can be seen in Fig. 7.

To orchestrate the co-simulation, the INTO-CPS tool[5] was used, which allows the user to import FMUs, connect their Inputs and Outputs, set parameters and co-simulate them. Both models were exported as FMUs, imported in INTO-CPS and the tank level and the valve control Inputs and Outputs were connected accordingly. To test the co-simulation, the tuning of the PID controller was done using the Ziegler-Nichols method[6] [12]. The value of the parameters of the controller's model `Kd` and `Ki` were set to zero and the value of `Kp` was increased until a stable and consistent oscillation was found, called critical gain `Kc`. Fig. 8 shows the results of the test tests `Kp` with values of 1500, 2000 and finally 2500 where the system shows a stable oscillation. The desired and actual tank level are shown where the Y-axis is height in meters, and the X-axis is the time in seconds. INTO-CPS allows the user to easily check all the values and times that were stored during the co-simulation, so the stored data revealed in Fig. 8.c that the period of the oscillation `Tc` at `Kc` was 0.6 seconds.

Following the Ziegler-Nichols rules for a basic PID controller knowing `Kc` and `Tc`, the actual values for `Kp` (1500), `Kd` (112.5) and `Ki` (5000) were calculated and set as the parameter to the FMUs of the controller. The co-simulation with the calculated values is shown in Fig. 9 where a stable and with minimum error response can be seen.

VI. Conclusion and Outlook

This paper presents a mapping between IEC 61499 models and the FMI, and an exporter of IEC 61499 models into FMUs. The mapping is done by finding the correlations between the data types of the two standards, and defining which parts of an IEC 61499 model could be offered to the FMI as internal variables, parameters and input or output. Since the execution of an IEC 61499 model is not defined in the

[4]https://www.openmodelica.org

[5]https://github.com/into-cps/intocps-ui

[6]http://www.mstarlabs.com/control/znrule.html

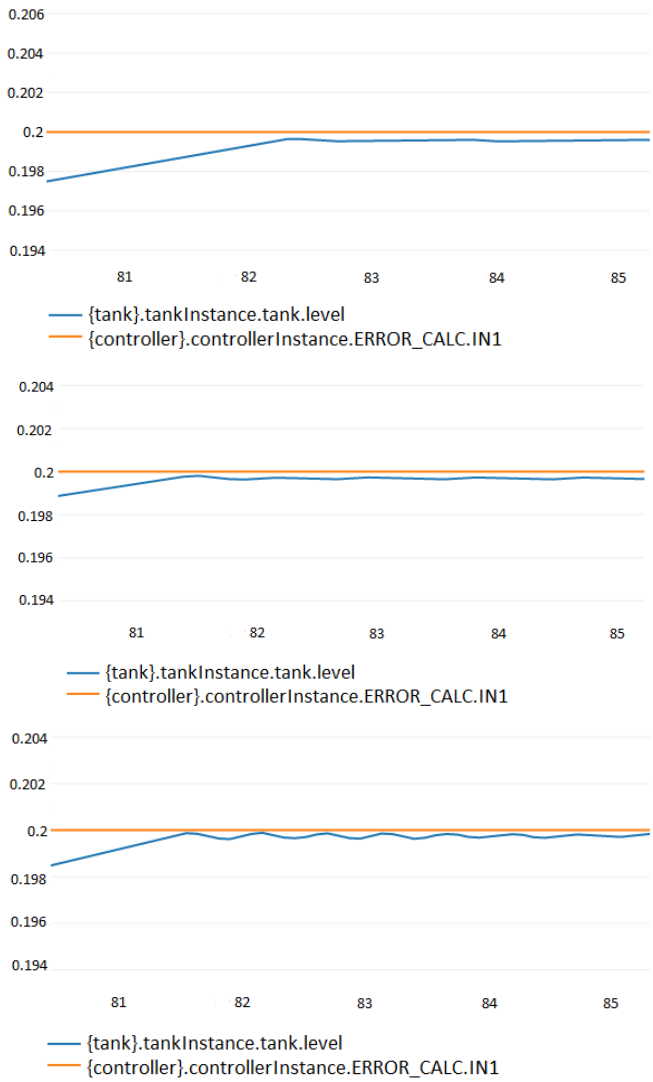

Fig. 8: Kp = 1500 (top); Kp = 2000 (middle); Kp = 2500 (bottom)

standard, specifications for it to support co-simulation using the FMI is presented.

The exporter was tested by modelling a PID controller of a tank system in IEC 61499, which then was exported as an FMU. A system of a tank was modelled using OpenModelica and also exported as an FMU. Both FMUs were imported in the INTO-CPS tool which allows connecting the interfaces of different FMUs. In the experiment, the control of a valve was offered as an output from the controller side, and the current tank level was the offered from the tank side. The controller also offered the constants of the PID controller as parameters, in order to allow the tuning of it.

The presented exporter opens the way to more complex co-simulations where more systems are involved. Vendors can offer their products as FMU with 3D visualization, and developers could test their controllers modelled in IEC 61499

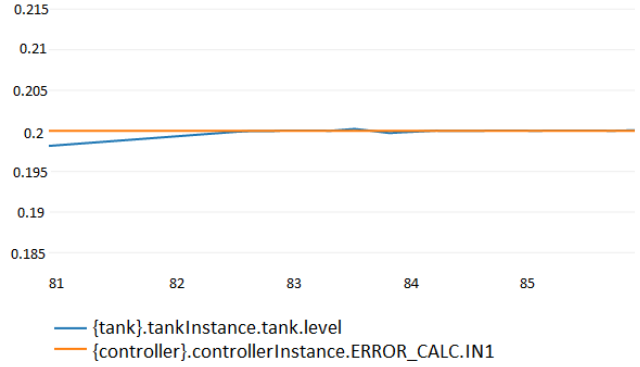

Fig. 9: Response with Kp = 1500, Kd = 112.5 and Ki = 5000

against it. Although more work is needed in investigating the simulation of the networking system that connects the different parts of the distributed systems.

Acknowledgment

This work is funded by the Electronic Components and Systems for European Leadership Joint Undertaking (ECSEL JU) under grant no. 737459 through the project Productive4.0 (https://productive40.eu).

References

[1] M. Hermann, T. Pentek, B. Otto. *Design Principles for Industrie 4.0 Scenarios*. 49th Hawaii International Conference on System Sciences, 2016.

[2] International Electrotechnical Commission. *IEC 61499-, Function blocks Part 1: Architecture*. Geneva, 2012.

[3] International Electrotechnical Commission. *IEC 61331, Programmable controllers - Part 3: Programming languages*. Geneva, 2003.

[4] J. Halmsjö, J. Fält. *Emulation of a production cell - Developing a Virtual Commissioning model in a concurrent environment*. [Online] Available: http://publications.lib.chalmers.se/records/fulltext/241210/241210.pdf. 2016, pp. 4-5.

[5] Modelica Association. *Functional Mock-up Interface for Model Exchange and Co-Simulation*. [Online] Available: http://fmi-standard.org/. 2014.

[6] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, Hans Vangheluwe, *Co-simulation: State of the art*, [online] Available: http://arxiv.org/abs/1702.00686, CoRR, vol. abs/1702.00686, 2017

[7] N. Pedersen, T. Bojsen , J. Madsen and M. Vejlgaard-Laursen, *FMI for Co-Simulation of Embedded Control Software*, The First Japanese Modelica Conference, Tokyo. [online] Available: http://www.ep.liu.se/ecp/124/010/ecp16124010.pdf, Linkoping University Electronic Press

[8] B. Wang and J. S. Baras, *HybridSim: A Modeling and Co-simulation Toolchain for Cyber-physical Systems*, 17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2013.

[9] J. Fitzgerald, P. Larsen, K. Pierce, M. Verhoef, and S. Wolff, *Collaborative Modelling and Co-simulation in the Development of Dependable Embedded Systems*, 26th Annual INCOSE International Symposium, Edinburgh, 2016.

[10] M. Spiegel, F. Leimgruber, E. Widl and G. Gridling, *On Using FMI-Based Models in IEC 61499 Control Applications*, Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), Seattle, 2015.

[11] J. Yan, V. Vyatkin, G. Weber, N. Beach, *Control and Hardware-In-The-Loop Simulation of Fruit Packing Machine with IEC 61499*, 10th IEEE International Conference on Industrial Informatics (INDIN), 2012.

[12] K. Aström and R. Murray, *Feedback Systems - An Introduction for Scientists and Engineers*, 2nd ed. Princeton, United States of America: Princeton University Press, 2008, pp. 303 - 312.