Standardized Integration of Real-Time and Non-Real-Time Systems: The Distributed Co-Simulation Protocol

Martin Krammer[1] Klaus Schuch[2] Christian Kater[3] Khaled Alekeish[4] Torsten Blochwitz[4]
Stefan Materne[5] Andreas Soppa[6] Martin Benedikt[1]

[1]VIRTUAL VEHICLE Research Center, Austria, {martin.krammer,martin.benedikt}@v2c2.at
[2]AVL List GmbH, Austria, klaus.schuch@avl.com
[3]Leibniz Universität Hannover, Germany, kater@sim.uni-hannover.de
[4]ESI-ITI GmbH, Germany, {torsten.blochwitz,khaled.alekeish}@esi-group.com
[5]TWT GmbH, Germany, stefan.materne@twt-gmbh.de
[6]Volkswagen AG, Germany, andreas.soppa@volkswagen.de

Abstract

Co-simulation techniques have evolved significantly over the last 10 years. System simulation and hardware-in-the-loop testing are used to develop complex products in many industrial sectors. The Functional Mock-Up Interface (FMI) represents a standardized solution for integration of simulation models, tools and solvers. In practice the integration and coupling of heterogeneous systems still require enormous efforts. Until now no standardized interface or protocol specification is available, which allows the interaction of real-time and non-real-time systems of different vendors. This paper presents selected technical aspects of the novel *Distributed Co-simulation Protocol* (DCP) and highlights primary application possibilities. The DCP consists of a data model, a finite state machine, and a communication protocol including a set of protocol data units. It supports a master-slave architecture for simulation setup and control. The DCP was developed in context of the ACOSAR project and was subsequently adopted by Modelica Association as a Modelica Association Project (MAP). It may be used in numerous industrial and scientific applications. The standardization of the DCP allows for a modular and interoperable development between system providers and integrators. In the end, this will lead to more efficient product development and testing.

Keywords: DCP, co-simulation, real-time, integration, standard

1 Introduction

Modeling and simulation represent key methods for successful development of cyber-physical systems. With the introduction of co-simulation methodologies, holistic cross-domain or system simulations became possible. This enabled exchange and integration of simulation models, tools, and solvers from different sources. The automotive industry is characterized by a multi-tiered organization. A deep hierarchy of suppliers performs distributed development and integration of automotive components, parts, and systems, that in the end are manufactured to complete vehicles. Depending on the stage of development, simulation models or real prototypes are available. The advantage of simulation models is that they can be tested in terms of software. Software tests are comparably cheap. However, they typically do not consider timing aspects or uncertainties of measured quantities. On the other hand, prototypes are advantageous when it comes to product validation. A prototype shows real-world behaviour and interacts with the environment. The disadvantages are that prototypes are usually very expensive, and safety critical or rare situations are difficult to test. For these reasons it seems advantageous to combine simulation and real-world prototype based testing approaches. For certain use cases this is considered as a possible solution to cope with the arising complexity, due to the high number of different scenarios and situations. This especially includes the field of automated driving (Doms et al., 2018). The European Union's automotive investment in research and development has increased to 53.8 billion Euro annually (European Automobile Manufacturers Association, 2018). Testing efficiency is key to successful product development. Interoperability of simulation tools and test infrastructure contributes to testing efficiency. Therefore the use of standards is essential.

The DCP (Distributed Co-Simulation Protocol) was developed in the ACOSAR project (Krammer et al., 2016). ACOSAR stands for "Advanced Co-Simulation Open System Architecture". ACOSAR was an ITEA 3[1] (Information Technology for European Advancement) project. Three original equip-

[1]http://www.itea3.org

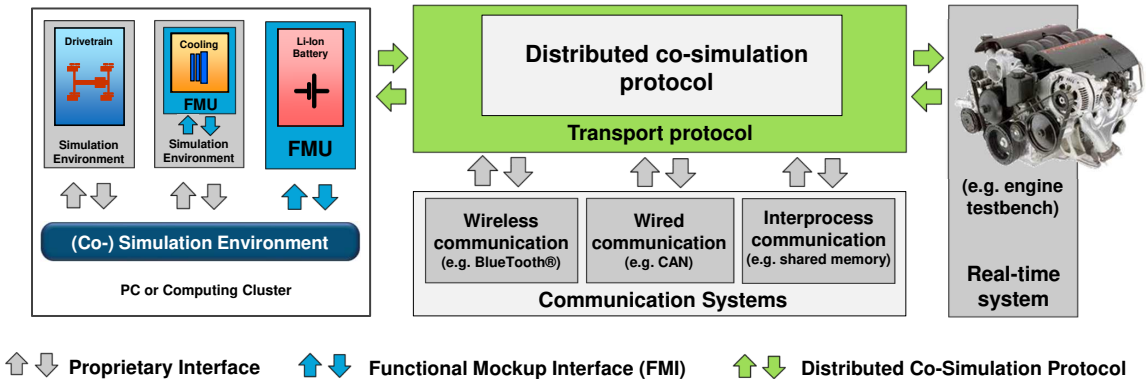

Figure 1. DCP concept.

ment manufacturers (OEM), 9 companies from the automotive supply chain, including simulation tool vendors, system and component providers, as well as 4 partners from research and academia cooperated. Their main goals were (1) the specification and demonstration of the DCP, and (2) preparation of standardization of the DCP with a recognized standardization body in order to promote it as the next co-simulation standard. Figure 1 shows an overview of the DCP's concept.

2 Related Work

The Functional Mock-up Interface (FMI) is introduced in (Blochwitz et al., 2011). The FMI was proposed to solve the need for interoperability between models and solvers. It was developed in the MODELISAR project, starting in 2008. The FMI specification is standardized as a Modelica Association Project (MAP). Its current version is 2.0 and was released in 2014. The FMI specification defines an interface for model exchange and co-simulation. Today more than 100 software tools support the FMI[2]. For distributed simulation environments, network communication technologies are frequently used in practice. However, such a "communication layer is not part of the FMI standard" (Modelisar Consortium and Modelica Association Project "FMI", 2014, p.93).

The Distributed Co-Simulation Protocol (DCP) is introduced in (Krammer et al., 2018). Its five main design ideas are highlighted; the improvement of interoperability between systems from different vendors, the integration of distributed real-time systems, the compatibility to a broad range of computing platforms, the support of multiple transport protocols, and development efficiency. The paper also introduces a typical architecture description of a DCP slave. It also describes the DCP's three different operating modes, namely hard real-time (HRT), soft real-time (SRT), and non real-time (NRT). They describe a DCP slave's relationship to absolute time. In general, deadlines must be kept for HRT and SRT operating modes. Simulation time must or should be synchronous to absolute time. The NRT operating mode can be used for distributed, computational co-simulation. In NRT operating mode, simulation time is independent from absolute time. The DCP specifies a state machine that governs the behaviour of a DCP slave. It defines five phases of a simulation cycle. Furthermore, the paper describes the main concepts of the communication protocol, including the design of protocol data units (PDU), the request and response mechanism, as well as the mechanism for configuration and exchange of input and output data. An example for UDP as a transport protocol is given, explaining the mechanism in detail.

In (Krammer and Benedikt, 2018) an algorithm for efficient generation of configurations for exchange of input and output data is given. The problem of finding such a configuration is an instance of the bin packing problem. In order to run such an algorithm, a co-simulation scenario description is required. The paper suggests a solution based on an XML schema description.

3 The Distributed Co-Simulation Protocol

The DCP is designed as a novel communication protocol on application level. It is intended for configuration and data exchange in co-simulation applications. The following sections provide details on features and technical novelties. Furthermore, the relationship to the FMI standard is highlighted.

3.1 DCP Feature Overview

3.1.1 Communication Architecture

The DCP implements the master-slave principle. It enables a DCP master to organize and configure its DCP slaves, so that a specific co-simulation scenario can be realized. A DCP slave represents a single subsystem of the co-simulation scenario. It can be a hardware-in-the-loop (HiL) system, a test bench, a

[2] http://fmi-standard.org/tools/

DOI 10.3384/ecp1915787

simulation tool, or similar system.

The DCP is a communication protocol intended for co-simulation configuration and data exchange. It is defined as a communication protocol that is independent of the underlying transport protocol. Classification of the DCP according to the Open Systems Interconnection (OSI) model (Zimmermann, 1980; International Telecommunication Union, 1994) is ambiguous. Its main properties fulfill major criteria for the application layer, e.g. access for application processes to the OSI environment. This is the highest layer defined in the OSI model. The DCP also features properties of the presentation layer, e.g. the design of DCP protocol data units (PDU), their associated fields and corresponding data types. The DCP implements a registration scheme, that allows the setup and simulation of co-simulation scenarios. This can be interpreted as a session. For the transport layer, the DCP defines mechanisms like the PDU sequence ID. Despite the fact that some transport protocols target properties like reliability (e.g. transmission control protocol, TCP), the DCP provides basic mechanisms to achieve similar behavior when a transport protocol is used that does not support this property (e.g. user datagram protocol, UDP).

3.1.2 State Machine

The DCP protocol is operated by a discrete state machine. The main design goal of this state machine is to ensure safe and reliable operation of real-time and non-real-time systems. In total, the DCP state machine consists of a set of 19 states grouped in 6 superstates. The entry point to the state machine is reached when the DCP software implementation is loaded to the DCP slave, the latter also indicates that the slave becomes available for registration by the master. A simulation cycle represents one complete pass through the DCP state machine.

The state machine enables simulation cycles having 6 different phases. In phase 1, a DCP slave is registered with a master which takes ownership of its registered slave. The later DCP slave is then exclusively controlled by its master. In phase 2, the DCP master configures its DCP slaves by generating a valid configuration scenario based on the DCP slave description of its slaves. Also for connection oriented transport protocols, a connection is established during the current phase. In phase 3, an iterative initialization process is carried out, the outcome of this process is establishing a consistent initial state over interconnected slaves. (see 3.2.2 for more details). In phase 4, The DCP slave in real-time operating modes is running and inputs/outputs are exchanged according to the configurations. Moreover, simulation time is mapped to absolute time. For non-real-time operating mode, simulation time does not progress at this phase. See section 3.2.3 for more details. Phase 5 applies only to non-real-time systems and each slave at this phase computes exactly one communication step and output is communicated to other slaves. Also the virtual simulation time is incremented by the number of specified steps. Phase 6 is intended to stop the simulation in a safe way, a stop of simulation can be triggered either by the master or by the slave itself.

3.1.3 Communication Protocol

To facilitate the communication between the master and slaves, DCP introduces the concept of Protocol Data Units (PDUs) that can be exchanged between the master and slaves. DCP addresses different types of PDUs which are used for different purposes and they serve distinct functionalities. So according to the functionalities of the PDUs, they are categorized in different families. DCP defines three top PDUs families named as Control, Notification (NTF) and Data (DAT) PDUs. The Control PDUs are further divided into Request and Response (RSP) families. Note that the Request PDUs are only sent by the master to its slaves and they consist of Configuration (CFG), State Change (STC) and Information (INF) requests. A slave upon receiving a request from its master has to acknowledge by sending a RSP PDU. DCP slaves can use NTF PDUs to inform the DCP master about certain events, for example, when the slave changes its state. Data PDUs can be used to transmit inputs and outputs between DCP slaves (slave-to-slave communication) and between the DCP master and its DCP slaves. Parameters (fixed or tunable), which are also packed in Data PDUs, can only be transmitted by the DCP master.

The Control PDUs are exchanged according to the request-response pattern. The latter pattern allows the DCP master to send specific requests to its slaves, it also enables each slave to inform its DCP master about the result of a requested action. Considering that DCP might be used on top of an unreliable transport protocol, packets loss might occur during the exchange of Control PDUs. Handling the latter situation can be determined by the DCP master and DCP slaves. For example, the DCP master might decide to initiate the retransmission of a Request PDU after a certain period of time.

3.1.4 DCP Data Exchange

DCP facilitates the exchange of input/output data between slaves. It enables a slave either to send data to other slaves directly or to send data to the master which passes this data on to all destination slaves. While the former communication way saves time and resources, the latter is intended for more sophisticated co-simulation configurations including extrapolation techniques or step-size control. A co-simulation DCP slaves configuration consists of a set of their DCP slave descriptions, the connections between their inputs and outputs as well as some other settings chosen by the master. This configu-

ration is rolled out to the slaves during the configuration phase. A slave that needs to send output data, receives a `CFG_output` PDU from the master, for each output data. The same applies to input data, the slave receives a `CFG_input` PDU for each input data it is going to receive. In addition to the two mentioned types of Control PDUs, the master also sends `CFG_target_network_information` and `CFG_source_network_information` PDUs. The latter two types of Control PDUs enable slaves to know where to send or from where to receive data, respectively, and their contents depend on the communication medium.

In addition to the input and output data, DCP also enables the master to send data for the parameters of its slaves and only the master can send this kind of data. Parameters can be either fixed or tunable, both types can be set during the configuration phase using the `CFG_parameter` PDU. While fixed parameters can be set only using the latter PDU, tunable ones can be set using the `DAT_parameter` PDU during any of the states that allow `DAT_input_output` PDUs to be sent. In the same way like the other Data PDUs, `DAT_parameter` PDUs are sent according to the stored configuration information which is received using the `CFG_tunable_parameter` PDUs during the configuration phase.

3.2 Technical Novelties

3.2.1 Integration Process

The DCP specification document describes the design of a DCP slave only. A DCP master is required to control a co-simulation scenario, which includes at least one DCP slave. In order to design and set up such a scenario, the DCP defines a non-normative default integration methodology. It defines the roles of a *DCP slave provider*, and a *DCP integrator*. The DCP integrator uses the DCP slave descriptions and a DCP master for configuration and control of the scenario.

The DCP slave description (DCPX) is a XML (Extensible Markup Language) file which describes one single DCP slave. It contains all static information related to one specific DCP slave. Its structure is defined by a normative XML XSD (XML Schema Definition) file. The top level structure of this schema definition file is shown in Figure 2. The DCP slave provider must provide an accompanying DCP slave description together with a DCP slave. The DCP master can attain all required information about available slaves by accessing their description files.

According to the specification, the DCP slave description must be stored in a single file named `dcpSlaveDescription.dcpx`, which in turn must be placed in a DCP file. The DCP file is a zip encoded

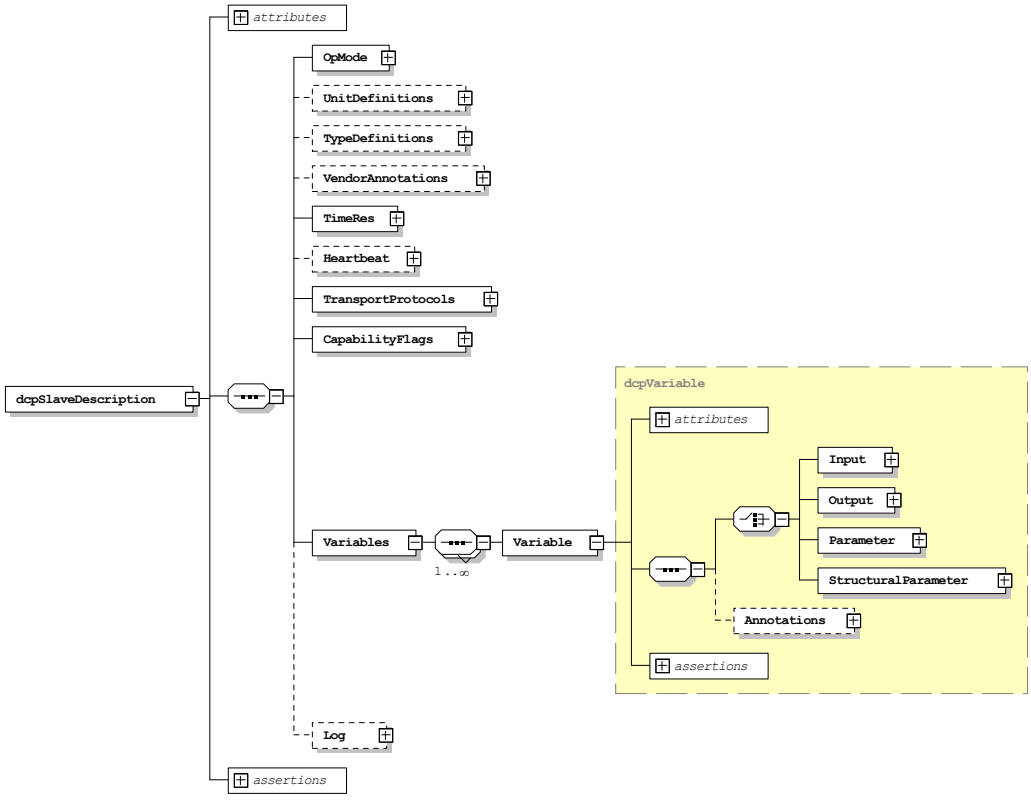

Figure 2. DCP slave description schema definition.

DOI 10.3384/ecp1915787

file (ISO/IEC JTC 1/SC 34, 2015) having the extension `.dcp`. Its internal structure is normative and designed to hold multiple DCPX files which are compliant to different DCP version numbers. This is one example of several design provisions taken into account to provide a future-proof DCP specification.

The set of DCP slave description schema files is normative. It does not only define the required structures of elements and attributes, but also supplementary assertions and constraints. Assertions and constraints are highly efficient for expressing logical relationships between elements and attributes.

Assertions are expressed in the `xs:assert` tag using the XML Path Language (XPath). An XPath expression addresses parts of an XML document in terms of a tree structure (Document Object Model, DOM). One location step in this tree consists of axis, node-test, and an optional predicate. An example for such an assertion is shown in Listing 1. It links the capability flag `canMonitorHeartbeat` to the defined XML child element `Heartbeat`. This prevents e.g. a set capability flag while the associated configuration information contained in the child element is missing. Assertions are a feature of XSD version 1.1. However, an XSL transformation (XSLT) file is specified, transforming the provided XSD version 1.1 schema definition file into a XSD version 1.0 schema definition file.

Furthermore, `xs:unique`, `xs:key` and `xs:keyref` tags are used to express constraints. Typical examples of application include the verification of uniqueness of names and the verification of cross-referenced key values.

In context of the DCP specification assertions and constraints provide strong formalisms which can be used for automated DCPX validation. This has shown to be advantageous in comparison to informal textual rules given in the specification document.

```
<xs:assert test="
((./CapabilityFlags/@canMonitorHeartbeat
    eq true()) and boolean(./Heartbeat))
        or
((./CapabilityFlags/@canMonitorHeartbeat
    eq false()) and boolean(./Heartbeat)
    eq false())
"/>
```

Listing 1. Assertion for capability flag and XML child element, as defined in the DCP slave description schema file.

3.2.2 Simulation Initialization

The DCP supports initialization calculations to achieve a consistent initial condition of connected DCP slaves. The DCP description file contains information about the DCP slave's dependencies. A dependency describes if an output is controllable by an input or parameter. Dependency information can be specified for the Initialization and Run superstates separately. The first is applicable prior to simulation, whereas the latter is applicable during simulation. Additionally, a DCP slave can mark outputs to be valid only in Initialization superstate. Such outputs are called *initial outputs*.

In the initialization phase simulation time does not progress. Hence, the master may roll out a configuration where the master receives all outputs and sends all inputs to the DCP slaves. The inputs sent by the master to the DCP slaves are not necessarily the outputs of other DCP slaves, a sophisticated master could send values chosen by a numerical solver instead (to solve algebraic loops). Algebraic loops in the context of FMI are explained in (Broman et al., 2013).

Connected DCP slaves may form pseudo algebraic loops. Such pseudo algebraic loops can be detected by exploiting the dependency information provided by the individual DCP slaves.

3.2.3 Simulation Synchronization

The master can observe the whole system to check if a global stable state was reached. The master informs the slaves afterwards to start the actual simulation test run. The achieved initial consistent configuration might still not correlate with reality. An output of a DCP slave could represent a physical quantity which typically fluctuates within certain boundaries. To minimize this difference and to circumvent this issue separate states were introduced. Each slave has the possibility to indicate that a local stable state has been reached, after fade out of transient oscillations. The master may observe the whole scenario to check if a global stable state was reached. If this is the case, the master may start the actual simulation run.

3.2.4 Connection-oriented Transport Protocols

The DCP supports connection-oriented and packet-oriented transport protocols.

To support connection-oriented protocols, two major mechanism were introduced to the DCP.

First of all, new states were introduced to distinguish between opening an endpoint and opening a connection. This is necessary to enable coordinated slave-to-slave communication. Without this distinction it would not be possible to detect if a slave has successfully opened its endpoints, ready to accept connections. Using this mechanism the master is able to instruct all slaves to open all endpoints first. After that, the slaves may establish their connections.

Second, the length of each PDU is sent on the stream, ahead of the actual PDU of the connection-oriented transport protocol. This eases implementation of slaves, because a slave is free to decide how many bytes he has to receive, independent from a

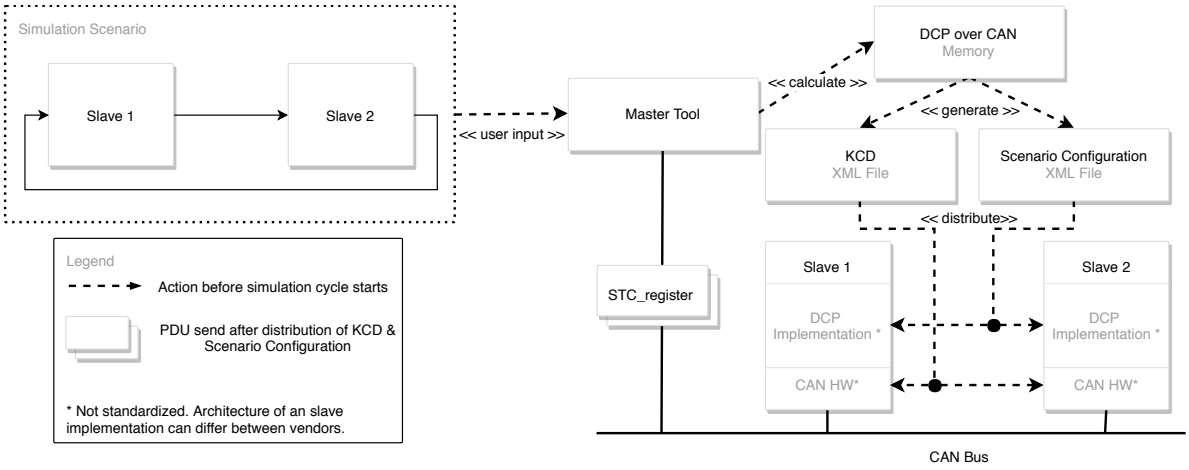

Figure 3. Possible scenario of DCP over CAN

slaves' configuration. In addition to that, PDU length verification also became possible.

Without the length ahead of the PDU a slave can only guess the length, based on its own assumptions. Misbehavior by other participants in terms of PDU length would not be detectable.

3.2.5 Non-native Transport Protocols

The DCP distinguishes between native and non-native transport protocols. Native DCP means that the mapping of PDUs to the transport protocol preserves the bit sequence.

If a transport protocol cannot fulfill this condition it is called a non-native transport protocol. One example of such a non-native transport protocol is the CAN bus communication system. Due to limitations of CAN, e. g. the CAN payload is limited to 8 bytes, not all Control PDUs can be send via CAN. For this reason the configuration of a slave will not be communicated by CAN.

To support the exchange of configuration PDUs for CAN an XML model is specified. The information contained in this model has to be generated by a master tool. It must be transmitted to the slave as a static configuration before simulation start. This model contains a K-matrix and the scenario configuration. The K-matrix contains all elements to describe the messages and signals of the CAN bus and the participation of the bus members to the messages. The scenario configuration contains all elements to describe the co-simulation scenario. When using a native DCP transport protocol instead, this information would be distributed to each DCP slave using configuration PDUs. In addition, the co-simulation scenario contains various other information, like DCP slave names, DCP slave identifiers, and their UUIDs (universally unique identifiers). The UUID is used to match information from these elements to DCP slaves. However, the way how information from the XML model is transferred to the DCP slaves is out of scope of DCP.

Figure 3 shows a possible scenario how DCP over CAN may be used in practice. A user defines the desired co-simulation scenario in master tool, supporting DCP over CAN. Based on this scenario the master calculates the K-Matrix and the scenario configuration in the DCP over CAN model and stores these information in different files. For the K-matrix e.g. the open source file format KCD was chosen. Any other file format describing CAN communications, e.g. DBC from Vector, would also be possible. After slaves are started, the CAN hardware is configured using the KCD file. The DCP implementation is configured using the scenario information. As a result, all slaves are waiting in state alive. The master tool sends out the register PDUs using the CAN bus and starts the simulation cycle.

3.2.6 Complex data types

New sensor technologies are currently evolving, for example camera, lidar or radar systems for the automotive market. In the automotive domain, these sensor types are used to enable advanced driver assistance systems (ADAS), to pursue the goal of automated driving. Test and operation of these systems rely on transmission of multidimensional or binary data types.

The DCP defines a binary data type to transmit arbitrary information. The binary representation consists of a 32 bit unsigned integer value that specifies the length in bytes of the actual data, followed by the binary data itself. The data is transmitted as given without any change in bit or byte order. Thus, the maximum length of data is limited to $2^{32}-1$ bytes. A DCP slave can limit this maximum length per variable, by specification of a maximum length in the DCP slave description. It is also possible to specify a MIME type compliant to RFC 2045 (Freed and Borenstein, 1996). The DCP integrator has to ensure compatibility between outputs and connected inputs

DOI 10.3384/ecp1915787

of binary data type, in the sense of maximum length and MIME type.

The DCP offers the possibility to define variables as arrays. An array variable is a data structure consisting of a collection of variables of the same type, each identified by an array index. A variable may have a constant number of dimensions. Each dimension has a size, defined by a constant or a structural parameter. By using a structural parameter it is possible to change the size of a dimension at any time.

3.2.7 Logging

The DCP supports the transmission of arbitrary log data from a DCP slave to its master. For that, it defines two different approaches, namely log-on-request and log-on-notification.

For log-on-request, log messages are stored by the DCP slave. They are picked up by the master on request and at any time. Thereby the master can avoid a high workload caused by log messages in the real-time-critical superstate Run. For log-on-notification, log messages are not stored within the DCP slave. Instead, they are transmitted to the master immediately. This mechanism supports devices with limited memory capacities, like micro-controllers.

The exact format of a log message is defined in the DCP slave description by using log templates. A log template consists of a category, level and a message. The category is defined in the DCP slave description. The possible values for the level are defined by the DCP. The category and the level can be used by the master to configure the logging of the DCP slave in a group wise manner. It is not necessary to configure every single log template individually.

The message of a log template defines the actual log string which is displayed to the user. In this string placeholders can be set, which define the values sent by a DCP slave to the master with the log message as seen in Figure 4. The full log message is then generated by the master, by replacing the placeholders with the received values from the slave.

3.3 Interaction with FMI

Right from the beginning of the ACOSAR project existing solutions for distributed co-simulation and system integration were carefully surveyed (Lichtenstein et al., 2016). Today, the FMI represents one of the most frequently used standards in the field of simulation. It is applied in many domains, including automotive, aerospace, maritime, or power grid domains. It is implemented in more than 100 commercial and open source tools. The ACOSAR consortium members recognized the feature set of FMI which represents the current state-of-the-art for co-simulation. As a consequence, the consortium proposed the adoption and extension of available concepts. The most important ones are described below.

The FMI follows a master-slave principle. In FMI for co-simulation different simulators can be coupled, if they are able to communicate data during simulation at certain time points. The master algorithm must handle data exchange between functional mock-up units (FMU) (Bastian et al., 2011). For example, it connects the output of an FMU to the input of another FMU. A co-simulation scenario represents a collection of interconnected FMUs. This introduces numerous challenges to the design of a master. The sequence of FMU calculations, or interpolation and extrapolation algorithms for FMUs operating with different step sizes represent some examples. A DCP master also connects the outputs of DCP slaves with the inputs of DCP slaves. In order to do so, a DCP master must be able to generate and roll out a configuration based on the intended simulation scenario (Krammer and Benedikt, 2018). In contrast to the FMI, the DCP also enables direct slave-to-slave communication. As an immediate consequence, dedicated coupling algorithms, like NEPCE (Benedikt et al., 2013) may only be applied if communication between DCP slaves is routed via the master.

The master-slave principle also follows economic goals. Slave providers agree on a standard, but com-

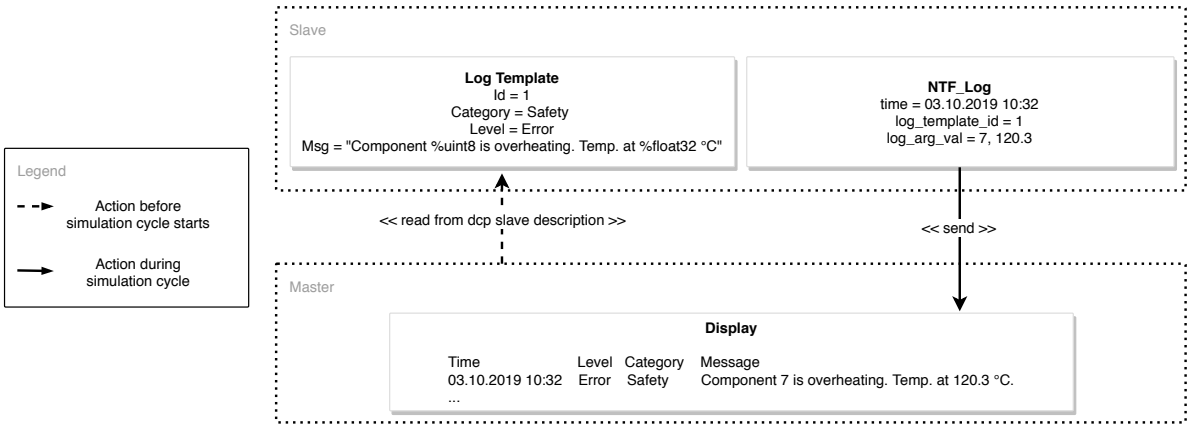

Figure 4. Example of log-on-notification mechanism.

pete in slave implementation. This allows an integrator to choose from best-in-class solutions. From a slave provider's perspective the market entrance barrier is lowered, since he is able to offer accessible solutions. Furthermore, the master algorithm, which is not standardized neither for FMI nor for DCP, may enable a stronger position on the market.

The FMI is operated using a state machine. Since state machines are one major method for the design and operation of communication protocols, the DCP was also defined on the basis of a state machine. The specification defines which PDUs can be sent and received in each state, the possible transitions between states, and the possible behaviour in each state. The DCP defines an `Initialization` superstate, which corresponds to the *Initialization Mode* of FMI.

The integration process of FMUs is supported by a standardized XML schema definition. It is used to generate one `modelDescription.xml` file per FMU. It contains the necessary information for instantiation and use of an FMU. It must be placed in the root directory inside an FMU, to allow an FMI master to read this information. Furthermore, a FMU may contain source code and/or compiled libraries. Due to the nature of DCP slaves, the inclusion of source code and/or compiled files within a DCP slave file is currently not explicitly specified.

4 Use Case

4.1 Overview

Typical use cases for the DCP include vehicle test benches, where real and virtual components are integrated into the same simulation scenario. This allows the execution of test cases that would not be possible in reality, due to cost, availability of components, or safety reasons. In this section we present a use case that is based on an engine testbed (PUMA from AVL List GmbH[3]) that interacts with a simulated vehicle and a simulated driver. A schematic overview of this use case is shown in Figure 5. The vehicle and the driver are simulated within one DCP slave ("Vehicle"), and the testbed available as another DCP slave ("Engine"). This use case is simulated as an SRT scenario. The connections of output variable to input variables between DCP slaves are shown as solid arrows in Figure 6.

4.2 Dependency Structures

The outputs of the DCP slave "Vehicle" are y_{torque} and y_{alpha}; the output of the DCP slave "Engine" is y_{speed}. The two DCP slaves are connected in the following

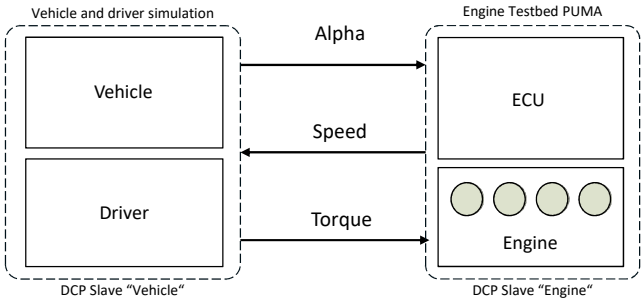

Figure 5. Vehicle-engine co-simulation use case.

way:

$$\text{Vehicle}.y_{\text{torque}} \rightarrow \text{Engine}.u_{\text{torque}}$$
$$\text{Vehicle}.y_{\text{alpha}} \rightarrow \text{Engine}.u_{\text{alpha}}$$
$$\text{Engine}.y_{\text{speed}} \rightarrow \text{Vehicle}.u_{\text{speed}}$$

To be able to start from a non-trivial start condition, both slaves declare parameters (Vehicle: $p^{\text{start}}_{\text{velocity}}$, $p^{\text{start}}_{\text{gear}}$, $p^{\text{start}}_{\text{alpha}}$; Engine: $p^{\text{start}}_{\text{speed}}$) that can be set in the `Initialization` superstate (see Section 3.2.2). In the `Initialization` superstate, the DCP slave "Vehicle" calculates the initial output as follows:

$$y^{\text{init}}_{\text{speed}} := f_{\text{speed}}(p^{\text{start}}_{\text{velocity}}, p^{\text{start}}_{\text{gear}}, \mathbf{p}_{\text{Vehicle}})$$

and the DCP slave "Engine" provides the initial output as follows:

$$y^{\text{init}}_{\text{alpha}} := f_{\text{alpha}}(u_{\text{torque}}, p^{\text{start}}_{\text{speed}}, \mathbf{p}_{\text{Engine}})$$

$\mathbf{p}_{\text{Vehicle}}$ and $\mathbf{p}_{\text{Engine}}$ are the vectors that contain all not explicitly mentioned parameters of the Vehicle and the Engine, respectively.

These initial outputs are used to set parameters ($\text{Engine}.p^{\text{start}}_{\text{speed}}$, $\text{Vehicle}.p^{\text{start}}_{\text{alpha}}$) of the opposite DCP slave:

$$\text{Vehicle}.y^{\text{init}}_{\text{speed}} \rightarrow \text{Engine}.p^{\text{start}}_{\text{speed}}$$
$$\text{Engine}.y^{\text{init}}_{\text{alpha}} \rightarrow \text{Vehicle}.p^{\text{start}}_{\text{alpha}}$$

If the master uses an output value of one DCP-slave to set a parameter of another DCP-slave, we call this a parameters connection. Such parameter connections are shown in Figure 6 as dotted arrows.

The dependency of outputs on other variables may be different in the `Initialization` superstate and in the `Run` superstate. In the `Initialization` superstate, the outputs of the DCP slave "Vehicle" are calculated according to:

$$y_{\text{torque}} := f_{\text{torque}}(p^{\text{start}}_{\text{velocity}}, p^{\text{start}}_{\text{gear}}, \mathbf{p}_{\text{Engine}})$$
$$y_{\text{alpha}} := p^{\text{start}}_{\text{alpha}}$$

The output y_{speed} of the DCP slave "Engine" in the `Initialization` superstate is determined by the parameter $p^{\text{start}}_{\text{speed}}$, i.e.:

$$y_{\text{speed}} := p^{\text{start}}_{\text{speed}}$$

[3] `http://www.avl.com`

DOI 10.3384/ecp1915787

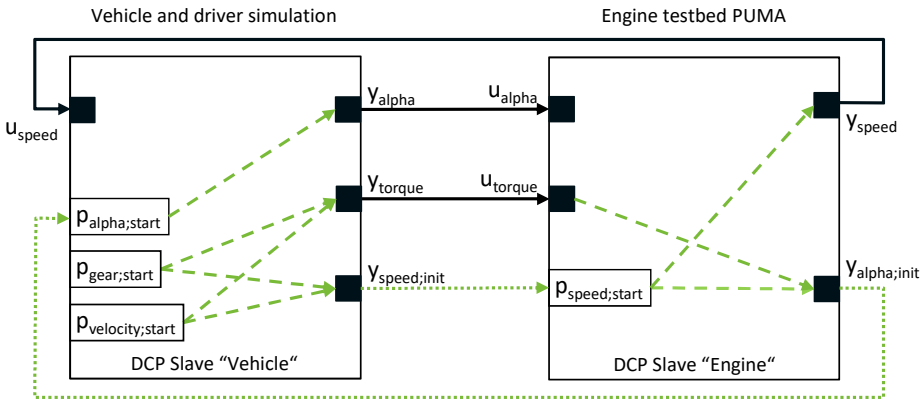

Figure 6. Vehicle-engine co-simulation scenario including dependency structure information during initialization.

4.3 Analysis

DCP slaves can provide information about the dependency structure of their outputs in the DCP slave description file (see Section 3.2.1). A DCP master may use this information to check if algebraic loops must be solved to achieve a consistent initial configuration. A graph may be used for such a check, where the nodes are variables of the DCP slaves. Each connection, parameter connection or dependency represents an edge of the graph. If the graph is acyclic, no algebraic loop needs to be solved. Note that without a given dependency structure, the DCP master would have to assume that each output depends on all inputs and parameters. The dependencies of outputs on inputs and parameters in the `Initialization` superstate of the described DCP slaves are shown in Figure 6. A dashed arrow from a variable x (an input or a parameter) to an output y indicates a dependency of y on x. It can be seen immediately that the graph does not contain any loops. Hence, a simple sequence of setting inputs/parameters after receiving output values is sufficient to achieve a consistent initial configuration. The DCP slaves state machines can subsequently be transitioned to superstate `Run` in order to perform synchronization (see Section 3.2.3) followed by the actual test case.

5 Standardized Solution

The Modelica Association[4] is a non-profit, non-governmental organization with members from Europe, North America, and Asia. Since 1996, its simulation experts have been working to develop the open standard Modelica and the open source Modelica Standard Library. Today it aims at coordinated standardization, development of software technology, and corresponding methods in the fields of cyber-physical systems and systems engineering. Currently the Modelica Association operates five Modelica Association Projects (MAP), where the DCP represents the most recent addition to the portfolio. The Modelica Association requires that all MAP results must be made available under an open source license.

The DCP was accepted as a MAP in 2018. The DCP specification document is initially published under a *Creative Commons Attribution Share-Alike 4.0* license[5]. The DCP slave description schema files, the DCP C++ reference implementation, and other supporting materials are initially published under a *BSD 3-clause* license[6].

MAP DCP follows its own rules. They are negotiated between its members and must be acknowledged by the Modelica Association. Contributions to MAP DCP are welcome. *Visitors* may contribute to MAP DCP in an informal way. *Advisory Committee* members actively support the design of the DCP. Its members must attend project meetings and sign a contributor's license agreement. They have access to development infrastructure, including mailing lists and file repositories. *Steering Committee* members have voting rights and define the strategy, feature roadmap, and future releases of the DCP. Furthermore, they must provide an implementation of the DCP specification, or part of it, in a commercial or open source tool. They should actively use DCP in industrial projects. Further information on these topics can be found on the DCP website[7].

6 Conclusion

The DCP enables integration of real-time systems and simulation environments in a standardized way. A stronger relationship between virtual and real worlds demands for new methodologies in simulation and test. Applications like automated driving, where high numbers of real world scenarios can be simulated before tests are conducted, can significantly benefit from the DCP.

[4] `http://www.modelica.org`

[5] `https://creativecommons.org/licenses/by-sa/4.0/`

[6] `https://opensource.org/licenses/BSD-3-Clause/`

[7] `http://www.dcp-standard.org`

The DCP specification version 1.0 is released by the Modelica Association. It represents the new state-of-the-art for co-simulation and test. The DCP is developed further by a consortium of original equipment manufacturers (OEM), simulation tool providers and software vendors, as well as suppliers for components and test equipment.

Despite the fact that the DCP was developed with other standards in mind, like the FMI, there are still challenges ahead. The FMI compatibility can still be improved, and the development of other software technologies like the SSP (System Structure and Parameterization) will require additional alignment activities in the future.

References

Jens Bastian, Christoph Clauß, Susann Wolf, and Peter Schneider. Master for Co-Simulation Using FMI. In *Proceedings of the 8th International Modelica Conference*, pages 115–120, 2011. doi:10.3384/ecp11063115.

Martin Benedikt, Daniel Watzenig, Josef Zehetner, and Anton Hofer. NEPCE - A nearly energy-preserving coupling element for weak-coupled problems and co-simulations. *International Conference on Computational Methods for Coupled Problems in Science and Engineering*, pages 1–12, 2013.

Torsten Blochwitz, Martin Otter, Martin Arnold, Constanze Bausch, Christoph Clauß, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, Manuel Monteiro, Thomas Neidhold, Dietmar Neumerkel, Hans Olsson, Jörg-Volker Peetz, and Susann Wolf. The functional mockup interface for tool independent exchange of simulation models. In *In Proceedings of the 8th International Modelica Conference*, pages 105–114, 03 2011. ISBN 978-91-7393-096-3. doi:10.3384/ecp11063105.

David Broman, Christopher Brooks, Lev Greenberg, Edward a. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Determinate composition of FMUs for co-simulation. In *2013 Proceedings of the International Conference on Embedded Software, EMSOFT 2013*, pages 1–12. Ieee, sep 2013. ISBN 9781479914432. doi:10.1109/EMSOFT.2013.6658580. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6658580`.

Thomas Doms, Benedikt Rauch, Bernhard Schrammel, Christoph Schwald, Edvin Spahovic, and Christian Schwarzl. Highly Automated Driving - The new challenges for Functional Safety and Cyber Security. White paper, TÜV Austria Holding AG and VIRTUAL VEHICLE, Vienna, Austria, 2018.

European Automobile Manufacturers Association. The Automobile Industry Pocket Guide 2018/2019. Technical report, European Automobile Manufacturers Association, Brussels, Belgium, 2018. URL `http://www.acea.be`.

Ned Freed and Dr. Nathaniel S. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045, November 1996. URL `https://rfc-editor.org/rfc/rfc2045.txt`.

International Telecommunication Union. Information technology – Open Systems Interconnection – Basic Reference Model: The basic model. ITU-T Recommendation X.200, International Telecommunication Union, 1994.

ISO/IEC JTC 1/SC 34. Information technology - Document Container File - Part 1: Core. Standard, International Organization for Standardization, Geneva, Switzerland, October 2015.

Martin Krammer and Martin Benedikt. Configuration of slaves based on the distributed co-simulation protocol. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 195–202. IEEE, 2018.

Martin Krammer, Nadja Marko, and Martin Benedikt. Interfacing Real-Time Systems for Advanced Co-Simulation - The ACOSAR Approach. In Catherine Dubois, Francesco Parisi-Presicce, Dimitris Kolovos, and Nicholas Matragkas, editors, *STAF 2016 Doctoral Symposium and Projects Showcase*, pages 32–39, Vienna, Austria, 2016. Dubois, Catherine Parisi-Presicce, Francesco Kolovos, Dimitris Matragkas, Nicholas.

Martin Krammer, Martin Benedikt, Torsten Blochwitz, Khaled Alekeish, Nicolas Amringer, Christian Kater, Stefan Materne, Roberto Ruvalcaba, Klaus Schuch, Josef Zehetner, Micha Damm-Norwig, Viktor Schreiber, Natarajan Nagarajan, Isidro Corral, Tommy Sparber, Serge Klein, and Jakob Andert. The distributed co-simulation protocol for the integration of real-time systems and simulation environments. In *Proceedings of the 50th Computer Simulation Conference*, SummerSim '18, pages 1:1–1:14, San Diego, CA, USA, 2018. Society for Computer Simulation International. URL `http://dl.acm.org/citation.cfm?id=3275382.3275383`.

Leonid Lichtenstein, Florian Ries, Michael Völker, Jos Höll, Christian König, Josef Zehetner, Oliver Kotte, Isidro Corral, Lars Mikelsons, Nicolas Amringer, Steffen Beringer, Janek Jochheim, Stefan Walter, Corinna Mitrohin, Natarajan Nagarajan, Torsten Blochwitz, Desheng Fu, Timo Haid, Jean-Marie Quelin, Rene Savelsberg, Serge Klein, Pacome Magnin, Bruno Lacabanne, Viktor Schreiber, Martin Krammer, Nadja Marko, Martin Benedikt, Stefan Thonhofer, Georg Stettinger, Markus Tranninger, and Thies Filler. Literature Review in the Fields of Standards, Projects, Industry, and Science. Technical report, ACOSAR Consortium, 2016.

Modelisar Consortium and Modelica Association Project "FMI". Functional Mock-up Interface for Model Exchange and Co-Simulation, Version 2.0, 2014.

Hubert Zimmermann. OSI reference model–The ISO model of architecture for open systems interconnection. *IEEE Transactions on communications*, 28(4):425–432, 1980.