 THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N° 332830 AND FROM SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.

CRitical S**YST**em Engineering **A**cce**L**eration

Heterogeneous Simulation Approach

D606.021

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Heterogeneous Simulation Approach
Deliverable No.	D606.021
Dissemination Level	CO
Nature	R
Document Version	V1.0
Date	2014-01-31
Contact	Thomas Kuhn
Organization	Fraunhofer IESE
Phone	+49 631 6800 2177
E-Mail	thomas.kuhn@iese.fraunhofer.de

AUTHORS TABLE

Name	Company	E-Mail
Thomas Kuhn	Fraunhofer IESE	thomas.kuhn@iese.fraunhofer.de
Sören Schneickert	Fraunhofer IESE	soeren.schneickert@iese.fraunhofer.de
Jean-Luc Johnson	EADS	Jean-Luc.Johnson@eads.com

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.1	20.09.2014	Outline	All
0.2	03.12.2013	Added FhG IESE context (Kuhn)	All
0.3	19.12.2013	Added content to chapter 3 (Schneickert)	All
0.4	20.12.2013	Added EADS content (Johnson, Kuhn)	All
0.9	27.12.2013	Added partner context (Kuhn)	All
0.91	15.01.2014	Internal Revision (Johnson, Kuhn, Schneickert)	All
1.0	29.01.2014	Processed comments from external Review (Kuhn)	All

CONTENT

Content of Figures

Content of Tables

1 Introduction

1.1 Role of deliverable

This deliverable collects the scientific results from WP 6.6. This first iteration presents results for bricks that have results already available. The other bricks from D6.6.1-1 are omitted in this deliverable iteration, and will be included in subsequent iterations of this deliverable.

Brick B2.47 (Approach for Integration of Heterogeneous Models) has a stronger focus on the integration of system models; its results would not be meaningful without their integration with brick B2.48 (Simulative Evaluation of Heterogeneous Models) that provides the core simulation environment. Brick B2.49 (Model-Based design Verification Method) provides the evaluation platform that triggers the simulation environment and interprets the results of the simulation.

1.2 Relationship to other CRYSTAL Documents

Deliverable D 6.6.1 describes the requirements for the bricks of this deliverable. Subsequent iterations of D6.6.1 will also contain evaluation scenarios and evaluation results. It also collects relevant related work, relevant research projects, and other relevant sources of information for WP 6.6 in general.

1.3 Structure of this document

This Deliverable is structured as following: Section 2 describes our progress in context of brick B2.47. Section 3 describes our progress in context of brick B2.48. Section 4 describes our progress in context of brick B2.49. Section 5 describes our progress in context of brick 2.40. Section 6 draws conclusions and lays out future work.

2 Approach for integration of heterogeneous models (B2.47)

2.1 Brick description

Specialized simulators simulate specific aspects of the real world with high accuracy; other aspects are approximated or ignored. The simulation of complex scenarios requires simulator coupling to address all relevant aspects in one semantically integrated scenario. In the domain of embedded systems, one common example scenario that requires simulator coupling is the simulation of shared control loops.

Today, in automotive, industrial, and avionic environments, control functions that control the same actuators are usually deployed on one electronic control unit (ECU) or computer, which is the hardware, the function is executed on. In the avionic domain, this is provided by the IMA platform that enables the segregated execution of mixed criticality functions on the same hardware. For the near future, plans exist to replace the common units with fewer multi-core CPUs. This way, independent functions for example would be deployed on the same hardware unit, while one function (i.e. one control loop) could be spread over several hardware units as well if this fits the system architecture. This way, the number of required hardware units could be drastically reduced.

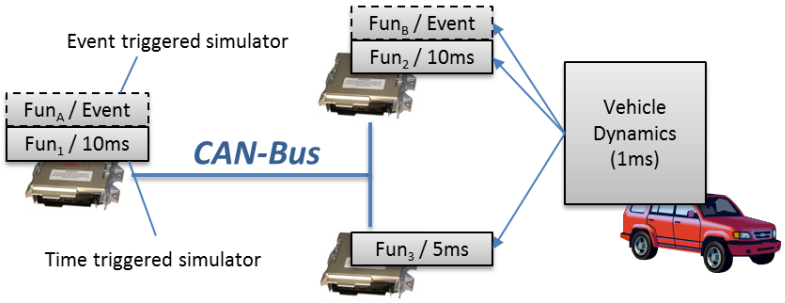

Figure 2-1: Example (automotive) scenario

Figure 2-1 illustrates an example of highly integrated automotive system architectures. It consists of two control loops, which are deployed onto multiple ECUs. Control loop 1 consists of periodically triggered functions Fun1, Fun2, and Fun3. Control loop 2 consists of event triggered functions FunA and FunB. Event triggered functions are commonly used for the handling of events that require timely processing. Periodic functions are executed at different periods; Fun1 is, for example, triggered every 10ms, while Fun3 is triggered every 5ms. Communication between ECUs is via an automotive network, e.g. via a CAN bus.To simulate all relevant effects, simulator coupling becomes necessary. In the described scenario, this includes the simulation of functional behavior, communication, and vehicle dynamics. Simulating all relevant effects by using specialized simulators separately is not sufficient, because effects often correlate with each other. For example, random delays of input data due to high network load and resulting jitter changes the functional behavior of algorithms. Only the combination of all relevant effects in one integrated holistic simulation scenario reveals all hidden interactions. Complex scenarios therefore need to be simulated by a combination of specializes simulators. Their integration requires the combination of multiple simulation models, which may be time based, event based, or based on finite state machine semantics.

2.2 Addressed requirements of Crystal USE-Cases

This brick mainly addresses the requirements of user story US 2.08: Multi-physics modelling and simulation. It is used in the context of the public aerospace use-case, as described in deliverable D208.010.

- Integration of heterogeneous simulation models and executable models
- Ability to integrate new model types and development environments when necessary
- Ability to add meta data to models, and to lookup models when instantiating simulation scenarios
- Ability to link simulation models across companies
- Confidentiality of company specific IP must be assured during simulation

2.3 Relationships to dependent bricks

Relationships exist to brick B2.48.

2.4 Solution approach

We propose to solve the challenge of integrating heterogeneous models by coupling individual simulation models through a framework that provides the semantic integration between the models, enables communication according to the Models of Communication and Computation of the individual models and that controls the simulation time, re-synchronization intervals and simulation accuracy of the resulting holistic simulation scenario.

Figure 2-2 illustrates a proposed coupled simulation scenario for the system shown in Figure 2-1. It consists of five time-triggered simulation domains that represent tasks that are running on the ECUs of the simulated system. Periodic tasks are triggered with three different periods. Additionally, event triggered Tasks (FunA, FunB) are part of the system; these tasks are triggered upon reception of a particular CAN message frame.

The vehicle dynamics simulator is included through a time triggered simulation domain as well. CAN bus simulation is realized via a CAN bus simulator (CANSim) in an event triggered domain; communication links exist to the time triggered domains that simulate functions of control loop 1 and control loop 2.

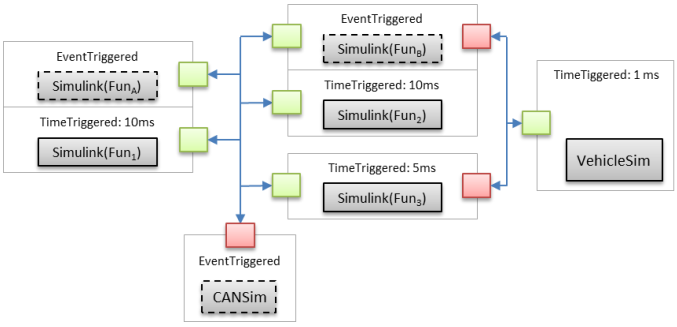

Figure 2-2: Simulation domains of highly integrated automotive architectures

Other simulation scenarios, e.g. from other application domains could be realized in a similar manner. Simulation scenarios are created out of components: Figure 2-3 illustrates the architecture of the simulator coupling framework “FERAL” – Framework for Efficient simulator coupling on Requirements and Architecture Level.

FERAL consists of runnable components, directors, and simulation components. Runnable components are the abstract base class of directors and simulation components; they are components of the simulation scenario. Ports and links represent communication between components.

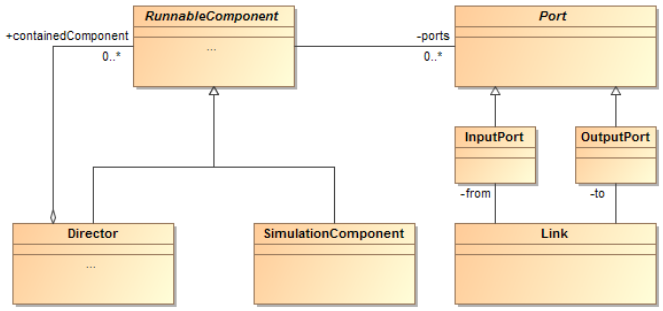

Figure 2-3: FERAL framework overview

Directors create simulation domains with defined semantics. Simulation domains of the FERAL framework manage the execution semantics of coupled simulators. They also manage the coupling of these simulators and synchronize communication between simulators. Simulation components are instantiated within simulation domains. In the simulation scenario shown in Figure 2-2, the Simulink component is a bridge to either a Simulink IDE debugging a Simulink model or to a compiled Simulink model. The CAN bus simulator CANSim integrates a CAN bus simulator component into the scenario.

Directors may be nested. A FERAL simulation scenario consists therefore of a tree of directors and simulation components. While directors control the semantics of the coupled simulation scenario; simulation components provide mostly syntactic interfaces to concrete simulators. A director controls four aspects that define the model of computation and communication of its semantic domain:

- Triggering of the activation of contained simulation components and controlling the active periods (i.e. periods of unsynchronized execution) of nested components.
- Controlling of the simulation time of contained simulation components
- Implementing the input port behavior of contained simulation components
- The times when transmitted messages waiting at output ports of connected components are forwarded to connected input ports of receivers.

Each simulation domain implements its local time. Horizontal synchronization between simulation domains is managed by the enclosing simulation domain. Vertical synchronization of nested simulation domains is provided by the FERAL framework. When simulation components are triggered, they are granted an active period. During this period, components may act independently of all other components. Synchronization with other components is performed at the end of the active period at earliest, depending on the simulation model of the enclosing director. Depending on synchronization intervals, time synchronization is more or less strict, enabling users to balance between accuracy and efficiency. In the shown example, the top-level simulation domain is time-triggered with a step size of 10ms. Therefore, local simulation times of all nested simulation domains must not deviate more than 10ms from each other. This also holds for nested domains and simulation components.

Execution of simulation components is completely controlled by enclosing directors. Nested directors define their own triggering semantics, which are mostly independently of the semantics of the enclosing director using the basic FERAL execution model. Directors however control the triggering of nested directors when necessary, e.g. for the simulation of finite state machines that enable or disable nested directors based on active states. Similar to simulation components, nested directors are bound by their active periods as defined by the enclosing director. This is necessary for maintaining synchronization across hierarchy levels.

Directors that are created within the FERAL framework are automatically bound to these basic semantics. New directors that are created by specializing existing directors automatically inherit this behavior; it cannot

be changed by specialized directors. Developers that create directors from scratch, e.g. because they need to implement exotic simulation models have the responsibility to ensure that their created directors conform to these basic semantics as well.

Data exchange between simulation components is managed through ports and links. Ports represent communication endpoints of directors and simulation components, links represent communication paths (cf. Figure 2-3). They connect simulation components within one simulation domain or simulation components across simulation domains. Simulated communication between simulation components of one simulation domain is controlled by the director of that domain. It controls the behavior of input ports and decides at which time data is forwarded from output ports to connected components. The time triggered director for example triggers all simulation components that are contained in its simulation domain periodically once per period. After all components have been triggered, it forwards data from all output ports to connected input ports. Data exchange between components therefore happens at the end of each period. Event triggered directors execute contained components, and immediately forward generated output messages to receivers.

Communication across simulation domains is managed by the FERAL framework as well. Like simulation components, directors define input ports as well. These are connected to input ports of nested directors or simulation components. Semantics of these input ports are managed by the enclosing director (cf. PortA in Figure 2-4). Input ports of directors are forwarded to input ports of contained directors or simulation components.

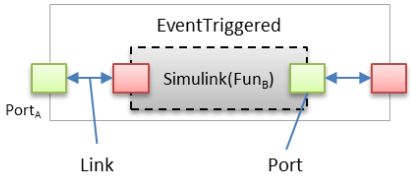

Figure 2-4: Communication across simulation domains

2.5 Relation to Crystal IOS

The proposed simulation coupling approach will be the foundation for the integrated execution of simulation models and functional mockup units under the control of the Crystal IOS platform. FERAL supports the coupling of fixed-step and variable-step solvers that enable the coupled simulation of functional mockup units. In addition, it is possible to integrate event-based and state-machine based models to support simulation of failure modes and communication models that simulate, for example, the failure of an individual sensor or of an individual de-icing device during flight.

OSLC (Open Services for Lifecycle Communication) will be used for service discovery and configuration of simulation scenarios that enables an integration of simulation services into the IOS platform, as well as the providing of simulation results via OSLC connectors. For the exchange of runtime data during executed simulations however, the runtime interface of FERAL will be used, because it is highly optimized for the support of distributed simulations and yields considerably less protocol overhead compared to OSLC.

2.6 Evaluation proposal

Evaluation scenarios are not defined yet. They will be finalized before the second iteration of this deliverable.

2.7 Related work

Simulator coupling is a topic of active research. Scientific literature already documents several couplings that were using specifically tailored interfaces. Additionally, related literature exists with respect to harmonized execution models and coupling frameworks.

One concrete simulator coupling is described in [1]. It illustrates a simulator coupling that was published already in 1997, which integrates the ANSYS and SABER tools for microsystem design. ANSYS is a simulation solution that implements algorithms for solving linear and non-linear engineering problems; SABER is a circuit and system simulator. This simulator coupling was realized for evaluation of thermal properties. It required semantic coupling of two different simulation models, which were realized in these two specialized tools. The authors of [2] describe the coupling of a simulator for netlists and another simulator for VHDL code. By coupling the simulators SLSim and SMASH, the authors enabled integrated simulation of continuous SPICE (Simulation Program with Integrated Circuit Emphasis) simulation models, as well as discrete VHDL models that enable the prediction of electronic circuits.

The work presented in [3] illustrates the networking domain as another application area of simulator coupling. By coupling simulators for link layer protocols and network layers, the authors build an infrastructure for locating interactions between effects on both layers. The work described in [4] applies the simulator coupling approach to the automotive domain for simulating the impacts of Car-to-X systems. This requires coupling of the OMNeT++ simulator, which provides a network simulation, and the road traffic simulator SUMO. The integrated simulation environment is used to evaluate the impact of Car-to-X solutions to vehicle behavior.

These works show that simulator integration is already today a proven technique that is applied whenever the coupling of simulation models becomes necessary. Due to the lack of coupling frameworks, however, development of such a coupling is time consuming, because interfaces and the overall simulation model are created individually for every coupling.

Modelisar [5] is a framework for the coupling of time-triggered simulators. It is based on a common simulation model that realizes time-triggered semantics and data flow communications. Several industry strength tools, for example, Simulink, and Modelica, implement Functional Mockup Interfaces (Modelisar/FMI) as defined by Modelisar. Modelisar, however, is focused on one simulation model; e.g. the integration of event-driven simulators that require simulation of queues is not possible. Therefore, simulation of complex networks, cloud-based services, and non-dataflow models in general are not in the scope of Modelisar.

SystemC/TLM [6] is another approach for the coupling of simulation models that is widely used in the domain of electrical engineering. It supports coupling of time and event triggered processes, but its semantics and simulation model are tailored to the simulation and coupling of components that simulate digital electronic circuits.

The aforementioned frameworks enable the coupling of simulation components, but cover only one simulation model. This limits their scope to one class of simulation areas. Modelisar for example is used in functional design, while SystemC is widely used in platform design. Development of integrated simulation scenarios requires coupling of simulators from different domains that use different simulation models, for example to reflect execution of functional designs on platform models.

Ptolemy II [7] is a framework that is used for exploring the integration of execution and communication models. It splits a system into different domains that are time-triggered, event-triggered, or that describe, for example, the triggering semantics of wireless networks. Each domain realizes one semantic model; Ptolemy defines an approach for the coupling of these domains. For our framework, we have adapted some aspects of Ptolemy and modified them to fit to the needs of a simulator coupling environment.

3 Simulative evaluation of heterogeneous models (B2.48)

3.1 Brick description

Nowadays real products are to be assembled from a broad spectrum of parts obeying a complex set of laws each. On a system and subsystem level these parts of a product mutually interoperate in extensive ways. Models representing such parts (and a combination of such parts respectively) can be used for the simulation of their interplay within controlled system environments thus simulating the behavior of the product to be evaluated. To this end simulating environments take a closed loop control based on events, time, and states to operate an arbitrary and often complex combination of simulation (sub) models.

In homogeneous environments this is realized by implementing the whole model and all its parts in one modeling language, but for different reasons this approach does not fit the needs:

- Parts of the model implementations to be evaluated exist already but may be the intellectual property of another party. Therefore they can't be used until the "wheel is re-invented".
- Already existing models, freely available but implemented in a different language cannot be used and therefore cause double work.
- Limitations of the used simulation environment force for re-modeling of available (sub) models.

Since products' parts are generally built by different producers each and every of the issues mentioned above may be valid for a simulation scenario, making evaluation unnecessarily expensive, and time consuming or even impossible. Therefore a simulation environment is needed that supports the deployment of heterogeneous models for heterogeneous simulation tools.

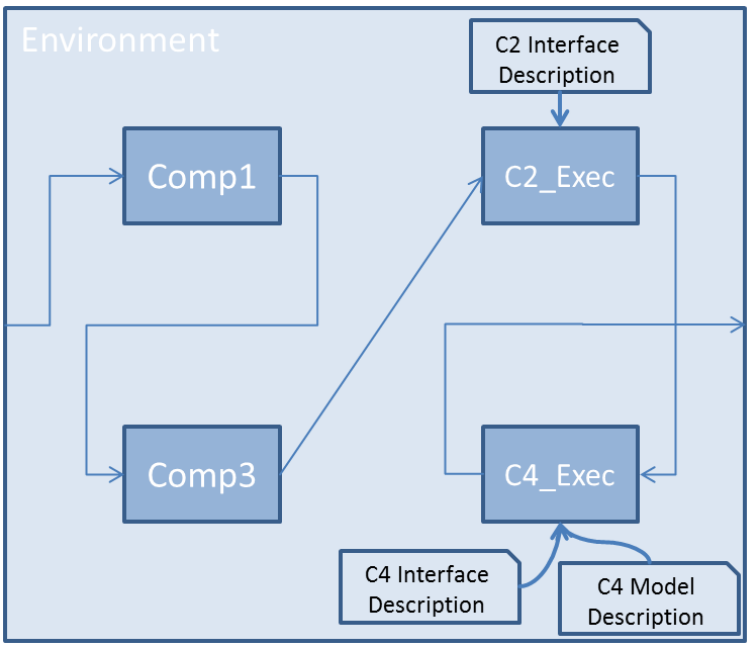

Figure 3-1: Heterogeneous scenario example

For the integration of heterogeneous structures into one's simulation environment two principle possibilities can be thought of. On the one hand executable part models must be able to integrate, i.e. when triggered with some input a correspondent output is directly provided (Co-Simulation). On the other hand non-

executable part models must be able to integrate by providing a model description that enables the environment to generate an executable (Model-Exchange).

3.2 Addressed requirements of Crystal USE-Cases

This brick mainly addresses the requirements of user story US 2.08. It is used in the context of the public aerospace use-case, as described in deliverable D208.010.

- Integration of heterogeneous simulation models and executable models
- Ability to integrate new model types and development environments when necessary
- Ability to add meta data to models, and to lookup models when instantiating simulation scenarios

3.3 Relationships to dependent bricks

Relationships exist to brick B2.47.

3.4 Solution approach

We propose to solve the challenge of a simulative evaluation of heterogeneous models by deploying model units like functional mock-up units (FMU) within a simulation framework that takes care of the interplay of the different FMUs deployed. By using functional mock-up interfaces (FMI) for co-simulation and model exchange it will be possible to build complex systems consisting of model parts based on a wide variety of simulation tools that already support FMI. While the simulation framework controls the process of deploying the contained heterogeneous models in a closed loop manner based on events, time and states part deployments are delegated to the FMUs that either execute their own simulation (co-simulation) or provide all data for the deployment of their underlying model within the specified simulation environment (model-exchange). As simulation framework we propose the usage of FERAL which provides the necessary domain and component types that are needed for a successful integration of FMUs. FERAL builds up simulation scenarios by combining and nesting simulation and directing components.

In a first step there is the need to adapt to FMU on a code basis as FERAL is Java-based and FMUs are C-based. To reach this goal we deploy JFMI (a Java wrapper for the FMI by the Ptolemy project). In a next step we have to build wrappers for FMI for Co-Simulation1 as well as for FMI for Model Exchange2 to translate the semantics of the FERAL framework to those of FMI.

[1] The intention is to provide an interface standard for coupling two or more simulation tools in a co-simulation environment. The data exchange between subsystems is restricted to discrete communication points. In the time between two communication points, the subsystems are solved independently from each other by their individual solver. Master algorithms control the data exchange between subsystems and the synchronization of all slave simulation solvers (slaves) [fmi-standard.org, 19.12.2013].

[2] The intention is that a modelling environment can generate C-Code of a dynamic system model that can be utilized by other modelling and simulation environments. Models are described by differential, algebraic and discrete equations with time-, state- and step-events) [fmi-standard.org, 19.12.2013].

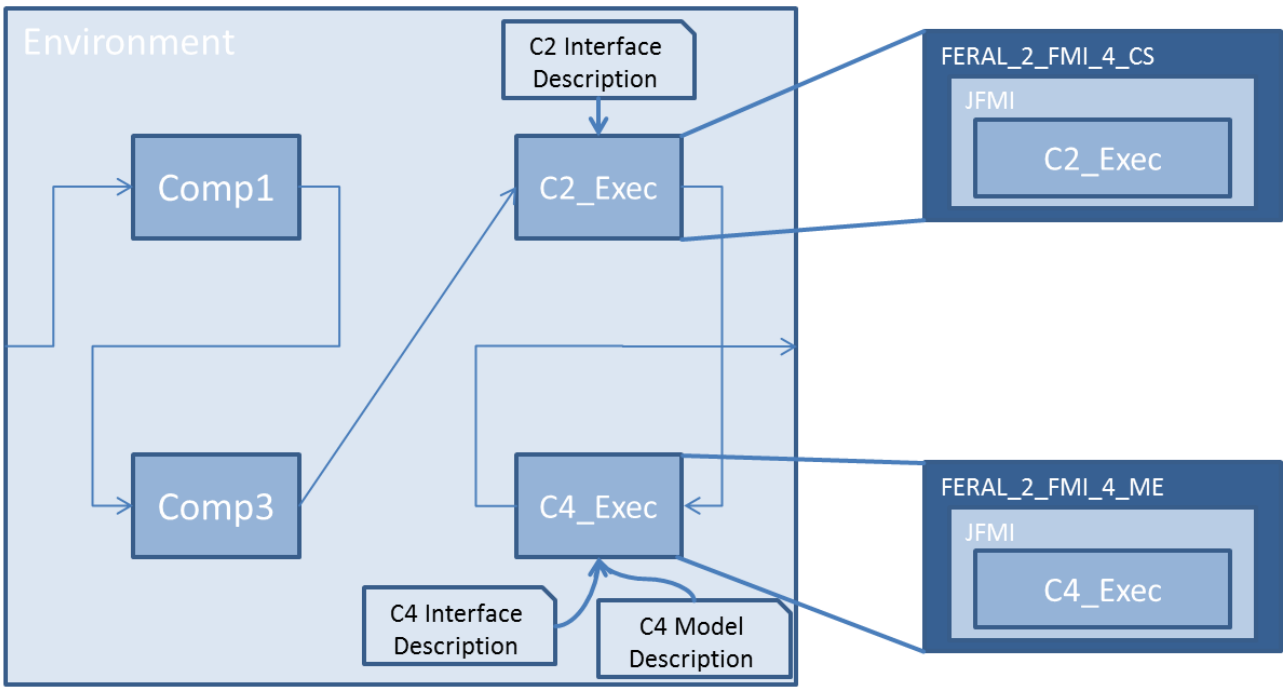

Figure 3-2: Heterogeneous scenario example – realization proposal

Signals handled within the FMI context, that is inputs, outputs, parameters, and status information as well as derivatives of inputs, and outputs w.r.t. time can be set and retrieved by the FERAL framework via the FMI wrapper. FERAL also counts for co-simulation algorithms and communication technology for distributed scenarios since FMI does not define these concepts.

Using FMUs in FERAL not only opens the framework for a variety of models already described by differential, algebraic and discrete equations, but supports also the integration of executable models from different tools supporting FMI and thus keeps intellectual properties secure and prepares distributed evaluation scenarios.

3.5 Evaluation proposal

Evaluation scenarios are not defined yet. They will be finalized before the second iteration of this deliverable.

3.6 Related work

FMI

Functional Mock-up Interface (FMI) is a tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of xml-files and compiled C-code. The first version, FMI 1.0, was published in 2010. The FMI development was initiated by Daimler AG with the goal to improve the exchange of simulation models between suppliers and OEMs. As of today, development of the standard continues through the participation of 16 companies and research institutes. FMI is supported by over 35 tools and is used by automotive and non-automotive organizations throughout Europe, Asia and North America.

2013-10-18: FMI specification v2.0 RC 1 released

https://www.fmi-standard.org/

Ptolemy

Ptolemy II is a framework that is used for exploring the integration of execution and communication models. It splits a system into different domains that are time-triggered, event-triggered, or that describe, for example, the triggering semantics of wireless networks. Each domain realizes one semantic model; Ptolemy defines an approach for the coupling of these domains.

The Ptolemy Project provides JFMI, a Java wrapper for the Functional Mock-up Interface (current version 1.0.2) that we use to adapt to the C-based FMUs.

http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm

4 Model-Based Design Verification Method (B2.49)

4.1 Brick description

Modelling and simulation of complex systems is at the heart of any modern engineering activity. Engineers strive to predict the behaviour of the system under development in order to get answers to particular questions long before physical prototypes or the actual system are built and can be tested in real life.

An important question is whether a particular system design fulfils or violates requirements that are imposed on the system under development. When developing complex systems, such as spacecraft, aircraft, cars, power plants, or any subsystem of such a system, this question becomes hard to answer simply because the systems are too complex for engineers to be able to create mental models of them. Nowadays it is common to use computer-supported modelling languages to describe complex physical and cyber-physical systems. The situation is different when it comes to describing requirements. Requirements are typically written in natural language. Unfortunately, natural languages fail at being unambiguous, in terms of both syntax and semantics. Automated processing of natural-language requirements is a challenging task which still is too difficult to accomplish via computer for this approach to be of significant use in requirements engineering or verification.

This brick will enhance the vVDR (virtual Verification of System Designs Against System Requirements) method (proposed in [1]) that enables verification of system dynamic behaviour designs against requirements using simulation models. In particular, it will show how natural-language requirements and scenarios are formalized using different tools and exported as FMUs and how simulation models can be composed automatically and used for design verification.

4.2 Addressed requirements of Crystal USE-Cases

This brick mainly addresses the requirements of use case 2.08:

- Integration of heterogeneous simulation models and executable models
- Ability to integrate new model types and development environments when necessary
- Confidentiality of company specific IP must be assured during simulation

4.3 Relationships to dependent bricks

Relationships exist to brick B2.47

4.4 Solution approach

The description of the virtual Verification of System Designs Against System Requirements method (vVDR) (see Figure 4-1) will be enhanced to take into account aspects that result from the fact of using FMUs instead of proprietary models (e.g. in Modelica or ModelicaML) and the need for integrating them on demand (e.g., perhaps even including the usage of IOS (see WP 6.1) for finding the appropriate set of models for the task at hand).

The enhancement will include the identification of meta-data necessary to enable an automated lookup and integration of models on demand (e.g. in order to perform verification of a particular design alternative against a set of formalized requirements).

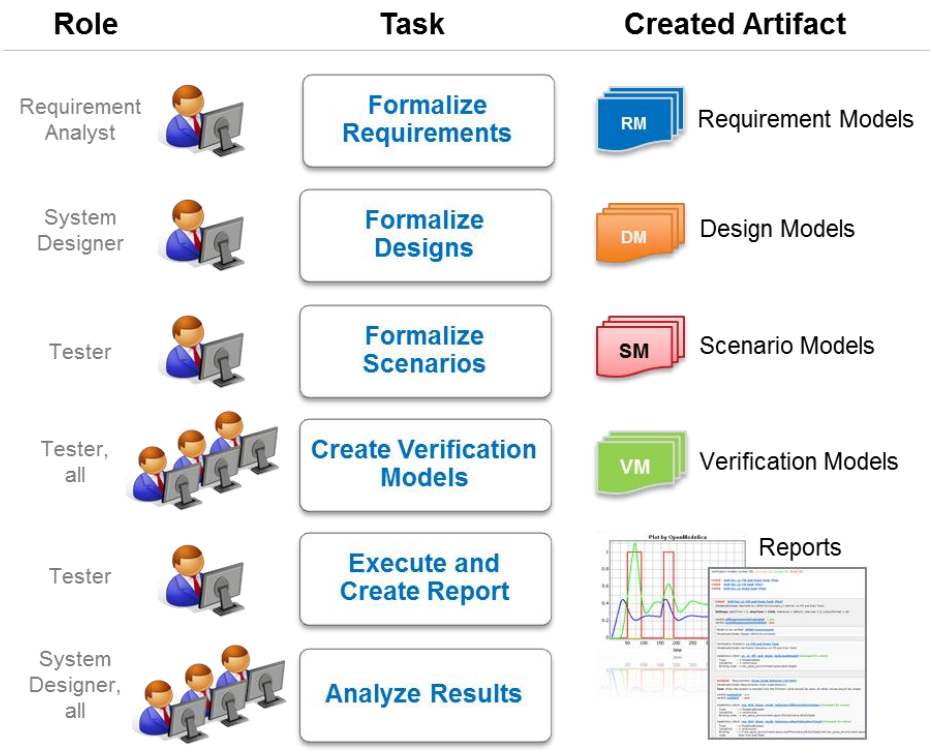

Figure 4-1: vVDR Method

The on-going work in this brick addresses the following questions:

- What is the minimum set of information to be captured in order to express that particular scenarios are appropriate for testing a certain design model against a set of requirements?
- What meta-data do we need to capture in order to enable lookup of models based on some input criteria? Is it possible to define generic search criteria, or are they all specific to the tasks on hand, How can we annotate existing models (FMUs) with meta-data?
- Having found the right set of models, what information needs to be captured in order to enable an automated integration of them (i.e., automatically connect models, or generate appropriate glue code)?

4.5 Evaluation proposal

Evaluation approach is not yet defined. It will be documented in the second issue of this deliverable after definition of appropriate use cases is in place.

4.6 Related work

The review of the state of the art shows that there is no approach that is specialized for design verification against natural-language requirements. For example, Model Based Systems Engineering (MBSE) methodologies focus more on the description of designs and do not provide means for formal verification of system design against natural-language requirements. Further, many of those methodologies use modelling to structure information and not to perform verification. Verification techniques have a differing focus or are limited in terms of model complexity. For example, model checking does not scale to cope with continuous-time models. Runtime verification primarily focuses on checking software runtime properties and addresses

challenges in generating efficient code from property monitors in order to minimize their impact while running them in parallel to the software being verified. Testing in general aims to find design errors by stimulating the system through appropriate test cases that include verdicts for determining the pass/fail criteria. The main intention of Model Based Testing (MBT) is to automate the generation of test cases based on a special white-box system model. Simulation is primarily used for predicting the behaviour of the real system that is represented by the model.

None of the listed approaches incorporates means for formalizing natural-language requirements for design verification using models. In conclusion, there is a need for a new approach which is tailored to a model-based verification of designs against requirements. The new approach should follow the heterogeneous simulation paradigm that underlies the WP 6.6.

5 ModelicaML Tool for Design Verification Based on Models (B2.40)

5.1 Brick description

The goal of this brick is to provide modelling and simulation environment for integrating requirement, design alternative and scenario models based on the FMI standard. This implies that models, i.e., FMUs, will be created in different tools (provided each tool used supports the FMI export). The ModelicaML tool will be enhanced to facilitate the usage and integration of FMUs in order to enable model-based design verification (see B2.49).

5.2 Addressed requirements of Crystal USE-Cases

This brick mainly addresses the requirements of use case 2.08:

- Integration of heterogeneous simulation models and executable models
- Ability to integrate new model types and development environments when necessary
- Confidentiality of company specific IP must be assured during simulation

5.3 Relationships to dependent bricks

The enhanced ModelicaML environment will be used to support the method developed in B2.49.

5.4 Solution approach

The main purpose of using ModelicaML tooling [5] will be for integrating models on demand (e.g., integrating the design alternative model with a scenario and a set of requirements). This will include creating new models (e.g. verification models, see [7]) that will be composed of the set of FMUs to be integrated.

The close link with Modelica allows reusing FMU import/export Modelica tool (e.g. OpenModelica [6]) functionality within ModelicaML. However, the open question in the on-going work is still whether a full import (i.e., translation) of FMUs into Modelica/ModelicaML models is necessary, or whether the required information can be captured using the data from the FMU description file (XML format).

Another challenge to be addressed is the format for storing the captured information. Candidates under investigation are: UML XMI format, a new XML format (based on a XML Schema), or the OWL format (i.e., storing data based on an ontology to be developed for that purpose).

5.5 Evaluation proposal

Evaluation approach is not yet defined. It will be documented in the second issue of this deliverable after definition of appropriate use cases is in place.

5.6 Related work

The main portion of the ModelicaML [5] concrete syntax is defined as graphical notation that facilitates different views (e.g. composition, inheritance, behaviour) on system models. It is based on a subset of the OMG Unified Modeling Language [1]. Since the ModelicaML profile is an extension of the UML meta-model, it can be used for modelling both with standard UML and with UML-based languages such as SysML [2].

Parts of the model are entered as text (i.e., Modelica equations or algorithmic code, modification and declaration expressions).

UML provides the modeller with powerful descriptive constructs at the expense of loosely defined semantics that are marked as “semantic variation points” in the UML specification. The intention of ModelicaML is to provide the modeller with powerful executable constructs and precise execution semantics that are based on the Modelica language.

Therefore, ModelicaML is based on a limited part of the UML meta-model and extends it using the UML profiling mechanism, offering new constructs to allow capturing of Modelica concepts that are not provided by UML and to support all concepts required for the vVDR method. However, like UML or SysML, ModelicaML is a graphical notation. ModelicaML models are eventually translated into Modelica code. Hence, the execution semantics are defined by the Modelica language and ultimately by a Modelica compiler that will translate the generated Modelica code into an executable form.

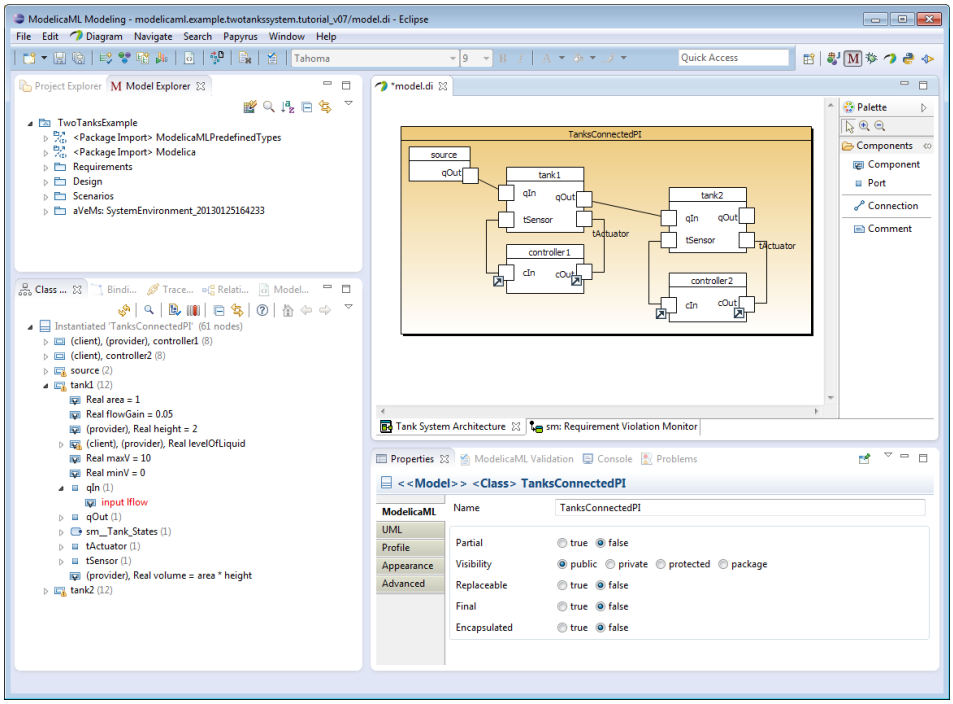

Figure 5-1: ModelicaML prototype GUI example

Along with the fact that ModelicaML integrates UML and Modelica (which addresses the need for an integrated modelling and simulation systems that contain both, hardware and software), it is designed to support the vVDR method. The ModelicaML tool (see Figure 5-1) extension (towards a heterogeneous simulation approach based on FMI) planned within this brick will widen the applicability of the proposed model-based design verification method and will contribute to improving the development process efficiency through automation.

6 Conclusion

The provided results indicate the progress that has already been made with respect to heterogeneous simulation. Due to necessary changes in bricks and a shift in focus, not all bricks could be addressed for this first deliverable, therefore, not all bricks are mentioned in this document. Those missing bricks will be added into the next iteration of this deliverable. Nevertheless, the results already indicate that a combination of Open Services for Lifecycle Collaboration (OSLC) for the configuration of simulation scenarios and the application of the functional mock-up interface FMI is a feasible way to realize heterogeneous simulation scenarios in the context of the CRYSTAL project.

7 Terms, Abbreviations and Definitions

Please add additional terms, abbreviations and definitions for your deliverable.

Table 7-1: Terms, Abbreviations and Definitions

CRYSTAL	**CR**itical S**YST**em Engineering **A**cce**L**eration
R	Report
P	Prototype
D	Demonstrator
O	Other
PU	Public
PP	Restricted to other program participants (including the JU).
RE	Restricted to a group specified by the consortium (including the JU).
CO	Confidential, only for members of the consortium (including the JU).
WP	Work Package
SP	Subproject
vVDR	Virtual Verification of System Designs Against System Requirements
CAN	Controller Area Network
FERAL	Framework for Evaluation of Systems on Requirements and Architecture Level
ECU	Electronic Control Unit, an automotive hardware platform
FMU	Functional Mockup Unit
FMI	Functional Mockup Interface
OSLC	Open Services for Lifecycle Collaboration
MBT	Model Based Testing

8 References

[1] OMG UML. (2011). *OMG Unified Modeling LanguageTM (OMG UML), Superstructure*. Retrieved Sept. 25, 2013, from www.omg.org

[2] OMG SysML. (2012). *OMG Systems Modeling Language (OMG SysML™).* Retrieved Sept. 25, 2013, from www.omgsysml.org

[3] Modelica. (2013). *Modelica specification, Version 3.3*. Modelica Association.

[4] Schamai, W. (2013). Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool.Ph.D. thesis.

[5] ModelicaML – A UML Profile for Modelica. http://modelicaml.openmodelica.org/

[6] OpenModelica Project. http://openmodelica.org/

[7] Schamai, W. (2013). Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool.Ph.D. thesis.