Communicating Process Architectures 2015
P.H. Welch et al. (Eds.)
Open Channel Publishing Ltd., 2015

Model-Driven Design of Simulation Support for the TERRA Robot Software Tool Suite

Zhou LU [a,1], Maarten BEZEMER [a] and Jan BROENINK [a]
[a] *Robotics and Mechatronics, CTIT Institute, Faculty EEMCS, University of Twente, The Netherlands*

Abstract. Model-Driven Development (MDD) based on the concepts of model, meta-model, and model transformation is an approach to develop predictable and reliable software for Cyber-Phsical Systems (CPS). The work presented here is on a methodology to design simulation software based on MDD techniques, supporting the TERRA tool suite to describe and simulate process communication flows. TERRA is implemented using MDD techniques and Communicating Sequential Process algebra (CSP). Simulation support for TERRA helps the designer to understand the semantics of the designed model, hence to increase the probability of first-time-right software implementations.

A new simulation meta-model is proposed, abstracting the simulation process of a TERRA model. With this new meta-model and our previously designed CSP meta-model, a simulation model can be transformed from its TERRA source. The Eclipse Modelling Framework (EMF) is used to implement the meta-model. The Eclipse Epsilon Framework includes the Epsilon Transformation Language (ETL) and the Epsilon Generation Language (EGL) are used for model-to-model and model-to-text transformation.

The simulation support is shown using an example, in which the generated trace text is shown as well. Further work is to implement an animation facility to show the trace text in the TERRA graphical model using colours.

Keywords. CSP, TERRA, simulation, model-driven development, meta-model, model transformation, Epsilon framework

Introduction

Context

In the traditional discussion about computing systems and physical systems, the cyber space and physical worlds are usually considered as separated parts [1,2]. While with rapid development of technologies and increasing need for information, the physical world is becoming more network and information oriented. This trend, the cyber space and physical world are becoming more closely integrated, and eventually the concept of Cyber-Physical Systems (CPS) was put forward [1,3,4]. Cyber represents for the information-based computation and network parts, including discrete computing processes, logical communicating processes

[1]Corresponding Author: *Zhou Lu, Robotics and Mechatronics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands*. Tel.: +31534894419; E-mail: z.lu@utwente.nl.

and feedback control processes; and the Physical part represents the processes, objects and events in natural or man-made physical systems, running according to laws of physics in continuous time. During the past decade, this CPS trend has already attracted a lot of interest from both academic and industrial sides; it even has been defined as the next computing revolution [4].

All in all, CPSes are becoming an integral part of modern society [5]. Examples of CPS include medical devices and systems, aircraft and air-traffic control systems, robotic systems, process control, factory automation. Most of these systems are known as safety-critical systems, which means the failures of these systems result in significant disasters or loss of human life. Thus, CPS must operate dependably, safely, securely, efficiently and in real-time [4]. However, the design of such complex embedded systems for safety-critical cyber-physical applications, in general needs too much time and is thus too costly. Also, there is a gap between the cyber part and the physical part during system integration in the end of the design phase, which will cause serious problems. Moreover, in general-purpose computing or embedded computing, certain key problems in CPS design rarely occur. For instance, the time to perform a task is an issue about performance, not about correctness. However, in CPS design, the execution time is normally critical to the proper functioning of the system.

In general, the design of CPS requires understanding the joint dynamics of computers, software, networks, and physical processes [6]. The combination of different research fields makes it quite challenging even impossible for professionals who only focus on their own domains. Generic multi-disciplinary methods, models and tools, which can deal with the complexities inherent to CPS and to guarantee robust operation of CPS under uncertainties are desired. We aim to develop methods and tools for model-driven cyber-physical co-design, which integrally treat the cyber and physical parts, to achieve first-time-right designs and eliminate the gap between cyber and physical parts during the early design phase, reducing time to market.

Related Work

Developing, modifying and integrating abstractions that cover *all* CPS design disciplines is one of the major challenges [7]. In control and systems engineering, abstractions are usually expressed using models. The increased use of design-automation tools has led to the introduction of a wide range of modelling languages, usually linked to specific analysis and simulation tools (such as Simulink [8], Modelica [9], and 20-sim [10,11]) or to various domain-specific standards (such as SysML).

Furthermore, during the last decade, driven by the increasing popularity of object-oriented programming languages, modelling has become the mainstream in software engineering as well [7]. Research directions, such as Model Driven Engineering (MDE) or Model Driven Design (MDD), and other model-based approaches in software engineering have led to the appearance of a new generation of computer-aided software engineering tools [12]. Consequently, merging the control, systems and software engineering built on the principles of MDD, in another way of speaking, designing CPS using MDD, has become one of the key research points.

Outline

Background information and motivation of this paper is introduced in the first section. Next, the simulation-support design methodology and meta-model implementation are described in Section 2. This is followed by use-case example to explain how to make use of the simulation support for TERRA. Conclusions and future work are given in Section 4.

1. Background

1.1. Modelling Cyber-Physical Systems

A complete model of a CPS represents the coupling of its environment, physical processes, and embedded computations [13]. Working with models has several advantages. Models can have formal properties, thus the model quality and consistency issues can be rigorously checked using formal verification tools [14]. Also, modelled systems can be tested and simulated off line [15], enabling developers to verify the logic of their application, to verify assumptions about its environment, and to verify end-to-end behavior [13]. These verification activities make system validation straightforward, even so that the real implementation gets right first time.

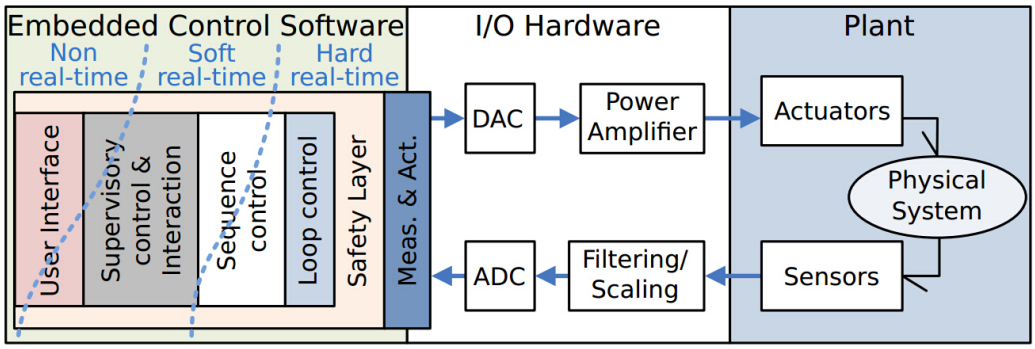

Figure 1. System architecture of a cyber-physical system. (Broenink et al. [16]).

In Figure 1, a typical system architecture of a CPS is shown, whereby the application area is robotics and mechatronics. Here, a CPS is divided into three parts: the Embedded Control Software (ECS), the I/O Hardware and the Plant. We mainly focus on the ECS part in this paper. The ECS can also be split into three layers with respect to different real-time properties.

Broenink et al. [17] presented an earlier version of a design procedure, which is used to develop ECS using MDD tools. Bezemer et al. [14] made further improvement to propose a way of working for ECS development using MDD techniques which helps to separate the different real-time layers as shown in Figure 1. For instance, a generic modelling tool can be used to construct the overall software architecture and connect the different building blocks with each other. Specific tools can be used to design the specific controllers, e.g. a loop controller can be designed using 20-sim. Figure 2 shows a generic concurrent design procedure for CPS using MDD [16].

Using MDD techniques, models of software architecture can be formally checked, combined with models of controllers, the possibilities of off-target debugging and (co)simulation can be emerged. Thus in further steps, a ”first-time-right” implementation, putting it in another way, high readiness level of codes of the embedded software, can be produced from models, which is also the ultimate goal of MDD. Apparently, in most situations a ”first-time-right” implementation is too ideal, however, MDD techniques can obviously help to reduce the amounts of on-target debugging efforts.

1.2. Concurrency in CPS

As mentioned above, models of software architecture need to be formally checked. Thus, those models must be formulated in a formal language. Furthermore, physical processes are compositions of many things occurring at the same time, so they are concurrent, which is

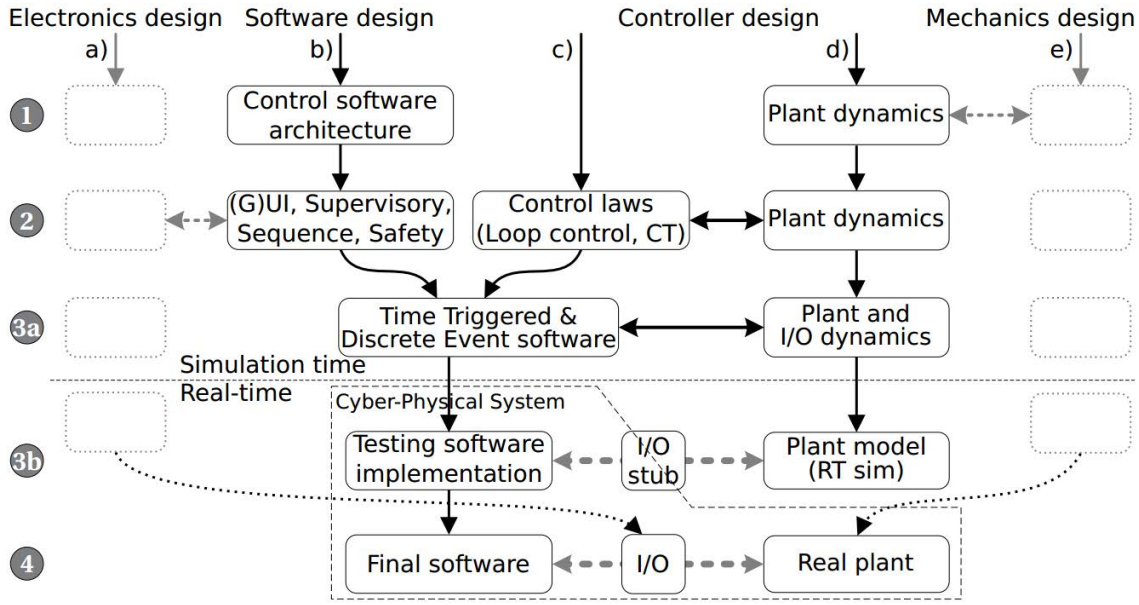

Figure 2. Design procedure for CPS using MDD. (Broenink et al. [16]).

extremely critical in CPS, unlike software processes which are deeply rooted in sequential steps. Measuring and controlling the dynamics of these processes is one of the main tasks of CPS design. Consequently, concurrency is intrinsicly present in CPS. Many of the technical challenges in designing and analysing embedded control software come from the need to bridge an inherently sequential semantics with an intrinsically concurrent physical world [6]. The Communicating Sequential Processes (CSP) algebra is a formal language for describing patterns of interaction in concurrent systems [18,19], which is a potential solution for formalization and concurrency issues in CPS.

1.3. CSP-based TERRA tool chain

Bezemer et al. presented an explicit meta-model based on the CSP algebra [20], which uses the Component-Port-Connector (CPC) meta-model [21] as its basis. It can elucidate CSP semantics and bring out many MDD possibilities, like (semi) automatic code generation, model-to-model transformation, etc. Besides this, a tool suite called Twente Embedded Real-time Robotic Application (TERRA) containing a graphical CSP editor is presented as well [20], which uses this CSP meta-model. TERRA provides graphical designs for embedded control software architecture, in which the designs are formulated as labelled and directed graphs, as well as model quality control and consistency checking.

Models are at the center of the TERRA tool suite. The TERRA tool suite overview is shown in Figure 3. A CSP model that conforms to the CSP meta-model can be graphically constructed using the TERRA CSP editor. Using a model-to-text (M2T) transformation, CSP machine readable (CSPm) files are generated, which can be formally checked by the Failure Divergence Refinement (FDR) tool [22]. From the CSP model, via M2T transformations, TERRA is also able to generate C++ source code for the LUNA framework [23], which is a hard real-time, multi-threaded, CSP-capable execution framework designed for embedded control software. After compiling the generated C++ source code together with the LUNA framework, the code can be run on a target (shown in Figure 3 as “LUNA Application”).

The CSP meta-models are implemented using the Eclipse Model Framework (EMF) [24]. Graphical Eclipse Framework (GEF) [25] is used for graphical model editor. The model validation, code generation and model transformation tools use the Eclipse framework called Epsilon [26]. In general, TERRA is an integrated collection of tools (Eclipse plug-ins) to support the design procedure shown in Figure 2.

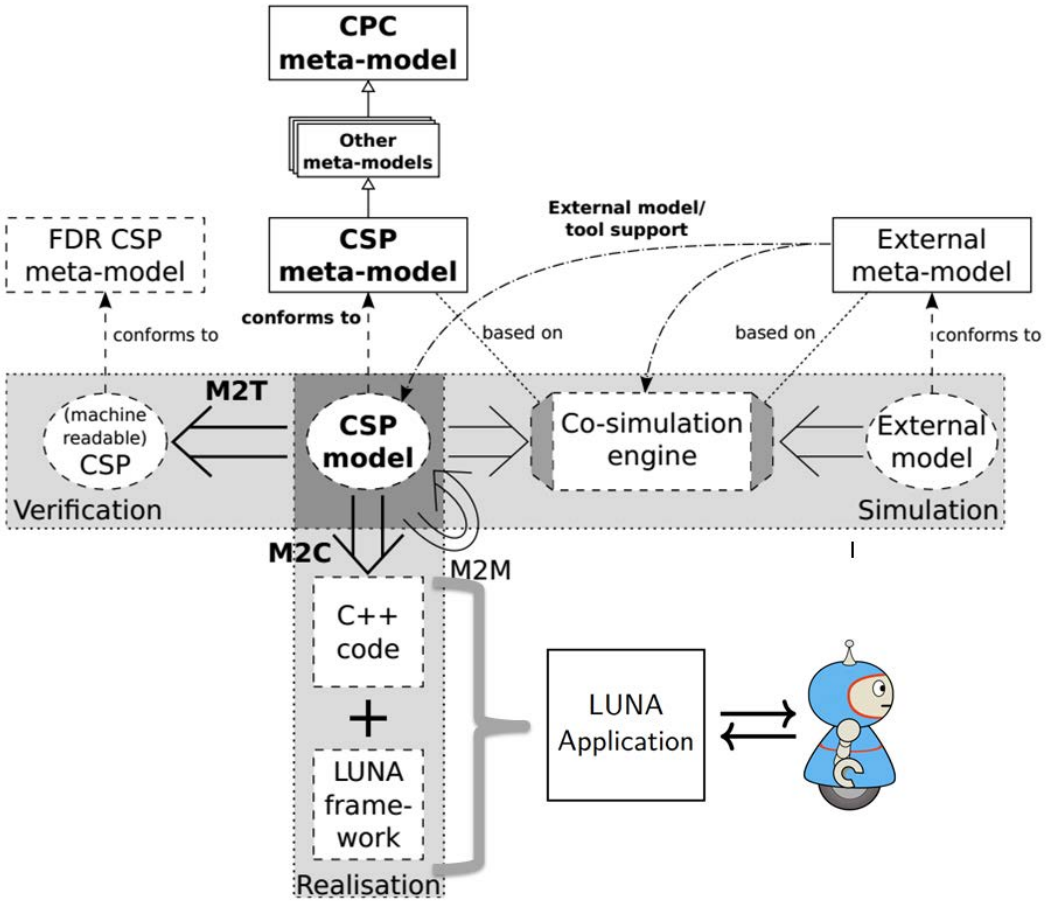

Figure 3. TERRA tool suite overview (Bezemer et al. [20]).

1.4. Motivation of simulation support for TERRA

As most CPS are safety-critical systems, executing unreliable software directly on an actual physical set-up might be hazardous. Also in some situations, the physical parts are not yet available. Thus, precise modelling for both the cyber and physical parts are required in CPS design as well as sufficient validation and verification of these models.

Simulations are commonly used in order to gain insight into models. These simulations can be categorised with respect to one or more different domain models. For instance, the Discrete-Event (DE) simulation for CSP models can determine the executing order of processes. In the Continuous-Time (CT) domain, plant dynamics can be modelled using bond graphs in 20-sim, and then be simulatied [11]. Combined simulations of multiple different types of domain models is so-called co-simulation. If the DE software and the CT dynamic model can be co-simulated together efficiently, relevant model-refinements can be done through (co)simulation results. Consequently, the reliability of software and the confidence of design will be both increased.

As mentioned above, the FDR tool can be used for formal verification of TERRA (CSP) models, among others on livelock and deadlock conditions. Thus, related software architecture design problems can be found and solved during the early design phase. However, the order of active processes, or in another way of speaking, the executing order of processes is not determined yet. For new model designers (students), 'executing' a CSP model can help them to understand the CSP semantics which is quite important in model design. Otherwise, even if the model structure is correct (without livelock or deadlock), the execution result might be different from expected due to the incorrect executing order of processes.

Obviously, well-designed simulation supports for model-driven CPS design is indispens-

able. In this paper, we mainly focus on simulation support for the TERRA tool suite with respect to the DE domain simulation.

1.5. Model-driven development for the tool chain

MDD uses model construction and model-based transformations to reach desired end-results, which in our case are the ECS for CPS. A tool chain like TERRA, designed based on MDD techniques, can contribute on automating software development, e.g., (semi) automatic code generation (M2T), preventing unnecessary human-based errors and reducing the time-to-market of software products. Meanwhile, in our case, a CSP model created by TERRA can first be formally verified and refined in structure abstract level before code generation, which will make it even closer to a "first-time-right" software implementation ideal.

Different types of models are used when a CPS is modelled. Each model type has its own advantages and disadvantages and is usable for a specific task or domain. Model transformations are used to transform these different types of models into a form that is compatible with all model types in order to combine them and reduce the gap between real-world problems and software implementation domains in the early design phase. For example, in the design procedure shown in Figure 2, a control model (control laws) can be designed in 20-sim which will be used to generate an XML file that describes the control model contents, and C++ source files which integrate the control algorithms inside will also be generated simultaneously. Then, the XML file can be transformed into a TERRA model which will be used as an external model by other upper-level TERRA models (represent the control software architecture) correspondingly.

As mentioned before, the CPC meta-model which provides elements like component, port and connection objects is the basis of the CSP meta-model. One or more other meta-models might serve as a base as well, but still they all are derived from the CPC meta-model in the end. Whether a meta-model is derived from one or more other meta-models depends on the situation and whether its functionality is shared by other meta-models [27]. Hence, derived from these meta-model theories, as shown at the right part of Figure 3, external meta-models can be introduced to provide means to store information about an external model and add support to the external tools. For instance, a 20-sim interface meta-model could extend the CPC meta-model and thereby seamlessly integrating 20-sim models into TERRA, and similar as hardware ports support described in [28].

The capabilities of using external meta-model and storing extra information about an external model in TERRA design schema, bring up the possibilities for our simulation support design. Same goes for the co-simulation engine, it does not need to know how to simulate the external model, only how to interact with the external tool to let it handle the simulation of the external model [27].

2. Simulation support for TERRA

2.1. Design methodology

Simulation is the imitation of the operation of a system or process, whereby in principle not all aspects are imitated. A preliminary model, often called a conceptual model that normally refers to non-executable higher-level abstraction of a system or process is needed first before simulation. It mainly represents the entity structure and composition of the system, as well as the relations between them. However, the conceptual model represents the system itself, whereas the simulation represents the operation of the system. To put it in another way, a conceptual model is in general not sufficient enough for simulation, and can not be simulated directly.

A simulation model is a representation of a system which can be simulated by means of experimentation [29]. Compared to the conceptual model mentioned above, it also represents the key characteristics, behaviours or functions of the system. A simulation model is executed on a simulation platform to generate simulation results, which is generally called as simulator [30]. In Figure 4, the basic concepts and relations in modelling and simulation are shown.

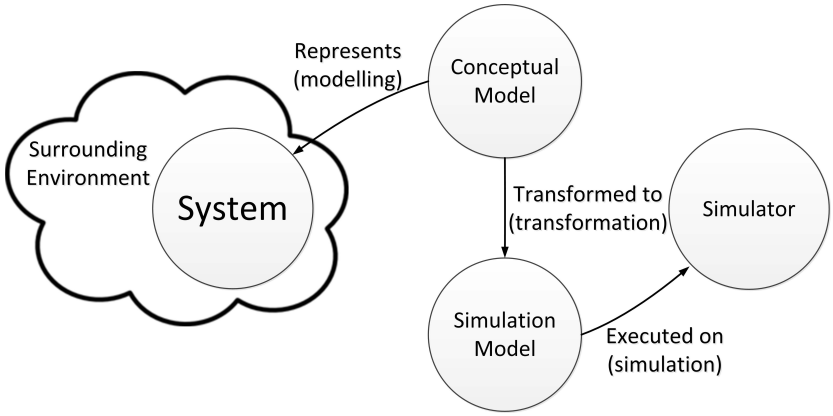

Figure 4. Basic Concepts and Relations in Modelling and Simulation.

A conceptual model also needs to be transformed into a simulation model by using some transformation rules or techniques. If a transformation method is formally defined and automatized, then a conceptual model can be transformed into an executable simulation model automatically based on these specific transformation rules or techniques.

In Figure 5, we show the structure of the entire simulation support system for TERRA indicating the design methodology and simulation working flow. The simulation support system mainly consists of two parts: an interpretor module and a simulator module. The left part of Figure 5 is the structure of the interpretor module. It is based on the OMG Meta-Object Facility (MOF) multi-layer specification [31]. The TERRA model conforms to the explicit CSP meta-model; the simulation model conforms to the simulation meta-model as we propose in this paper, and which will be introduced later. These two meta-models both conform to the Ecore meta-model, which makes it possible to transform from one to another. Since TERRA is designed using MDD techniques based on the explicit CSP meta-models, so the definitions of all TERRA model elements are known already. Thus sets of Epsilon transform rules for transforming a TERRA model to a simulation model can be implemented, which conform to both of the CSP meta-model and the simulation meta-model. The transformation rules need to be able to determine the order of components that needs to be simulated, using the domain-specific details of the meta-model.

In our case, the TERRA model is the conceptual model that describes the ECS component structure and composition as well as the relations among them. As shown in Figure 5, after M2M transformation (Step A), the automatically generated simulation model is used for M2T transformation (Step B), which generates a simulation trace text that will be used by the simulator module as additional execution context of the simulation model. Combining the simulation model and the corresponding trace text, the simulation engine can broadcast the resulting stream of events (events queue) to interact with the user (Step C or D) or for simulation animation using (Step E).

2.2. Simulation meta-model

A meta-model is a model of a model. A model conforms to its meta-model in the way that a computer program conforms to the grammar of the programming language in which it is

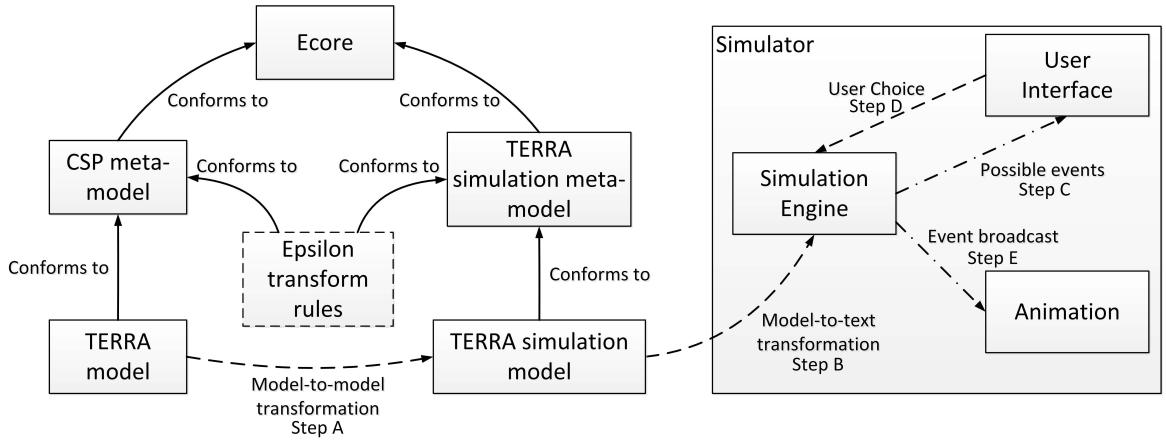

Figure 5. Design Methodology of Simulation Support for TERRA.

written. Meta-models can be considered as an explicit description: as constructs and rules of how a domain-specific model is built.

As mentioned in previous sections, the CPC meta-model captures the common factors used in component-based designs. It provides means to design component-port-connection like models, as it provides elements like component (CPCModel), port (CPCport) and connection (CPCconnection) objects, as shown in Figure 6. It also shows how the CSP meta-

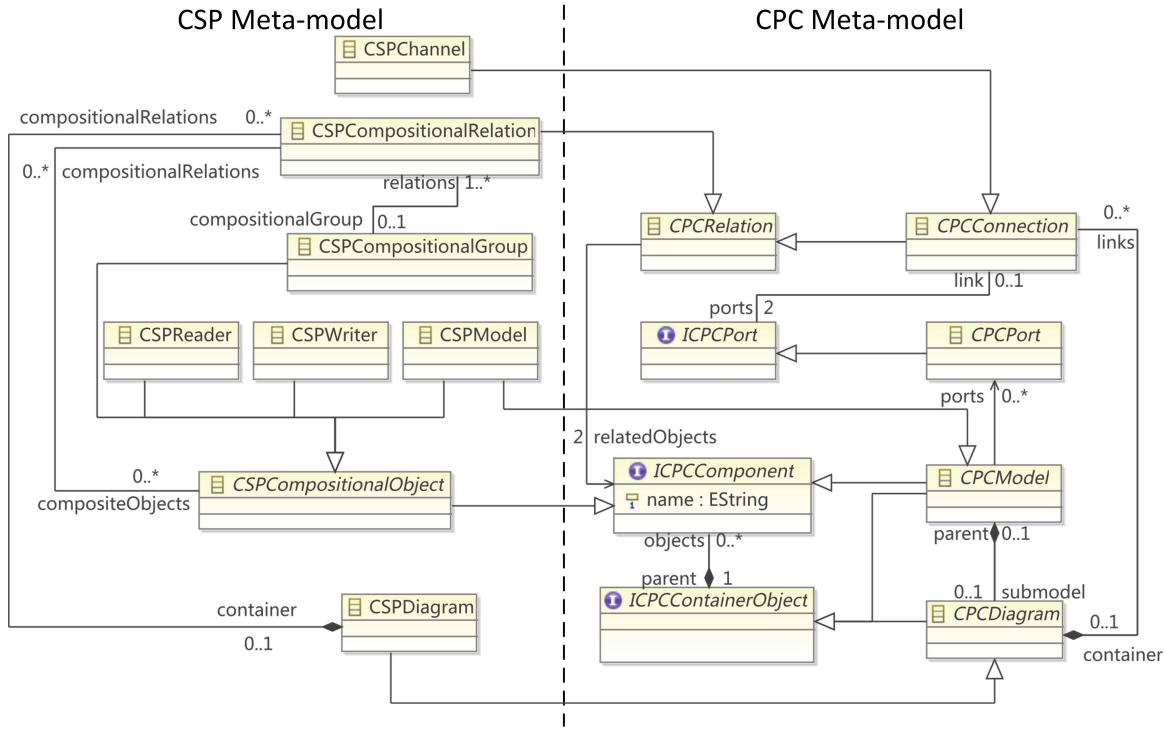

Figure 6. Partial CPC and CSP meta-model, showing the relations between them.

model is derived from the CPC meta-model and extending it. As we can see from the CSP meta-model diagram, it includes CSP domain-specific elements like writer (CSPWriter), reader (CSPReader), channel (CSPChannel) and objects compositions (CSPCompositional-Group and CSPCompositionalRelation) etc., to provide means to the TERRA model with respect to the CSP algebra. However, for simulation purposes, these CSP meta-model components are not sufficient enough. The 'executing' order of model elements is invisible directly from the TERRA model. Therefore, we need to find a suitable and efficient way to determine

the order of the active model elements and make the conceptual TERRA model to become an executable simulation model.

By extending the CSP meta-model, more additional facilities could be added for simulation purposes. However, it might break the existing composition structure of the meta-model, which can bring unpredictable problems to the later C++ code generation. Another disadvantage is that the extended meta-model can only be used for a specific type of simulation. A specific simulation rule (engine) has to be defined for each model that conforms to different meta-models, thus there is no re-usability at all for other kinds of CPC-based models. For instance, the architecture model which is able to describe the relations between different TERRA models and the interaction with external model or hardware ports outside TERRA. Thus, we have to redo the extending procedure for each CPC-based meta-model and reconsider the code generation issue as well, which is quite inconvenient and inefficient. Similar disadvantages occur if CSPm, i.e. machine-readable CSP would be used. First, CPSm is short of some C++ code-generation specific information, like guard expressions, variable and data type descriptions; it can only be used as supplementary or assistance for simulation. Although CSPm is generated based on the CSP algebra from TERRA models and can be interpreted to determine the 'executing' order, for each CPC-based meta-model, code generation algorithms need to be re-created which is obviously a drawback from MDD perspective.

Due to the disadvantages mentioned above, we create a new simulation meta-model next to the CSP meta-model to add simulation support for the TERRA tool suite according to the simulation methodology and work flow as we proposed. Note that the current CSP meta-model structure, composition and relations among them, stays the same. The simulation meta-model abstracts the simulation procedure with respect to the TERRA model. It is divided into three main meta-classes, or we can say three abstraction levels, see Figure 7.

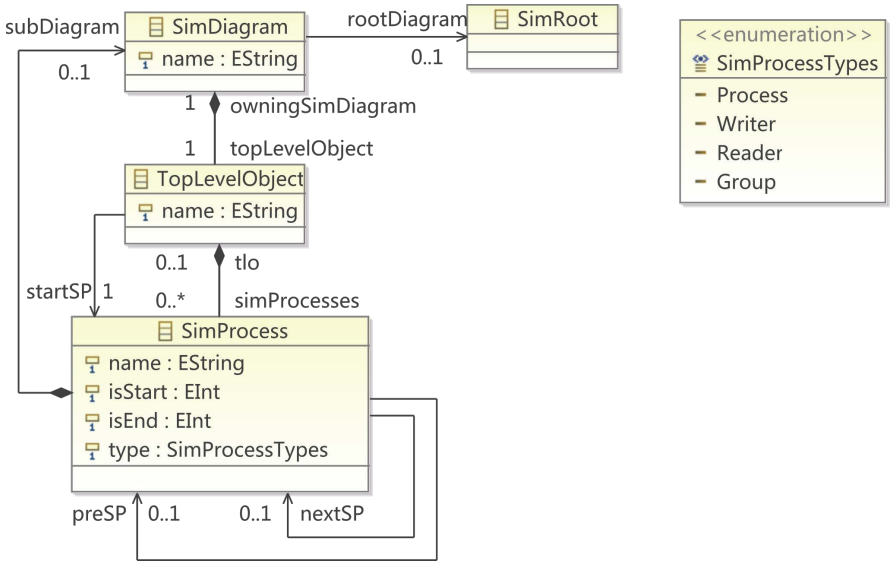

Figure 7. Simplified simulation meta-model.

On the first level, a SimDiagram represents a root CSPDiagram or a submodel level CSPDiagram. It provides means to the starting point of the whole simulation procedure as well as the sub-start point of each submodel level CSPDiagram. An intermediate meta-class is called as TopLevelObject. In most cases, it represents CSPCompositionalGroup, which is used for grouping a series of processes that have the same compositional relation type. It can also represent CSPModel, CSPWriter or CSPReader depending on specific situations when there is no CSPCompositionalGroup object defined. So the transformation rules conform to both the CSP meta-model and the simulation meta-model, and can be used to determine the executing order of model elements. On this level, the transformation rules need to have fa-

cilities to determine the first simulation process. We have implemented this determination using a (pseudo) random selection algorithm to deal with the Parallel syntax in CSP algebra, which will be introduced later. On the third level, a SimProcess represents the CSP elements, which are defined as objects in CSP meta-model, like CSPModel (process), CSPWriter, CSPReader, CSPCompositionalGroup, etc. Furthermore, the executing order of each model element can be determined and described with the help of the transformation rules and the preSP and nextSP association relations. In Table 1, the mapping representation between CSP meta-model and simulation meta-model is shown.

Table 1. Mapping Representation between CSP Meta-model and Simulation Meta-model.

CSP meta-model definition	Simulation meta-model definition	additional attribute(s)
`CSPDiagram`	SimDiagram	*name* *topLevelObject*

`CSPCompositionalGroup`
if CSPCompositionalGroup is not defined
`CSPModel`
`CSPWriter`
`CSPReader`
	TopLevelObject	*name*
		OwningSimDiagram
		startSP

`CSPModel`
`CSPWriter`
`CSPReader`
`CSPCompositionalGroup`
	SimProcess	*name*
		isStart
		isEnd
		type
		preSP
		nextSP
		tlo
		subDiagram

```
A: Start from a SimDiagram;
Find top level object;
B: Find a following SP;
if(can not find a following SP)
        if(can not find a parent diagram) end;
        else go to parent diagram level->B;
else continue;
if(currentSP does not contain subDiagram){
        if(currentSP is not end){
                simulate currentSP;
                go to currentSP->nextSP;
        }
        else{
                simulate currentSP;
                go to B;
        }
}
else go to A;
```

Listing 1. Pseudo code for the simulation rules (engine).

With the simulation meta-model, the disadvantages mentioned before can be removed or reduced. Still taking the architecture model as an example, the architecture meta-model is also derived from the CPC meta-model. It is easy to re-create the transformation rules for architecture to simulation model transformation, as most of the operations/functions in the transformation rules can be re-used. Besides this, there is *no* need to modify or change the source meta-model nor the code-generation parts. Another benefit is that the simulation rule (engine) need only be designed for the simulation model. So if another kind of model is transformed to a simulation model, the simulation rule (engine) can be used as a generic one. In Listing 1, we present the pseudo code for the simulation rules (engine), SP stands for simulation process.

2.3. Implementation

The simulation meta-model is implemented using EMF, the same as the CPC and CSP meta-models. The EMF is an Eclipse framework providing means and edit facilities to create a specific meta-model. It uses a meta-model itself called Ecore as mentioned before, to model that specific meta-model. Ecore is also the meta-meta-model of the CPC, CSP and the simulation model, as shown in Figure 8.

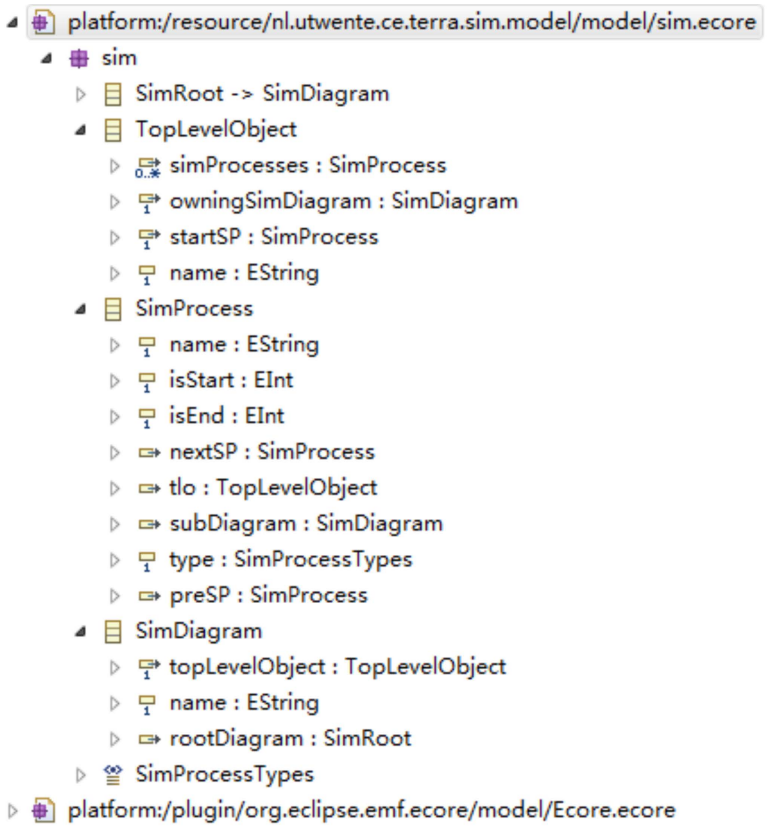

Figure 8. Simulation Meta-model Implementation in EMF.

The TERRA-to-simulation model transformation and simulation trace text generation use the Eclipse Epsilon framework, the same as the 20sim-to-TERRA model transformation and C++ code generation. The Epsilon framework provides a language called Epsilon Object Language (EOL), used to consolidates some common MDD facilities as a base language. Like the Epsilon Transformation Language (ETL) for model-to-model transformation and the Epsilon Generation Language (EGL) for code (text) generation, and several more languages for model validation and comparison. The hierarchical architecture provides means and possibilities for us to re-use the operations/functions defined in EOL to reduce tedious

and costly re-coding work as mentioned before when defining multiple transformation rules for different source and target meta-models. In Figure 9, the overview structure of the Epsilon framework is shown.

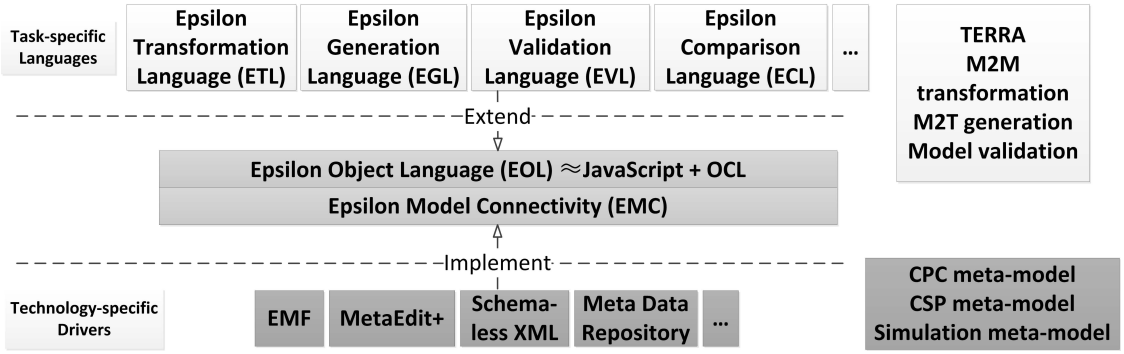

Figure 9. Overview Structure of Epsilon, modified from (Kolovos et al. [32], Epsilon website [33]).

3. Example

The example shown in Figure 10 is a simple TERRA model with all CSP constructs described in Table 1. The model contains Sequential and Parallel relations, sub-model diagrams and a rendezvous channel with a writer and a reader. Based on this example, we explain how to make use of our simulation support for TERRA.

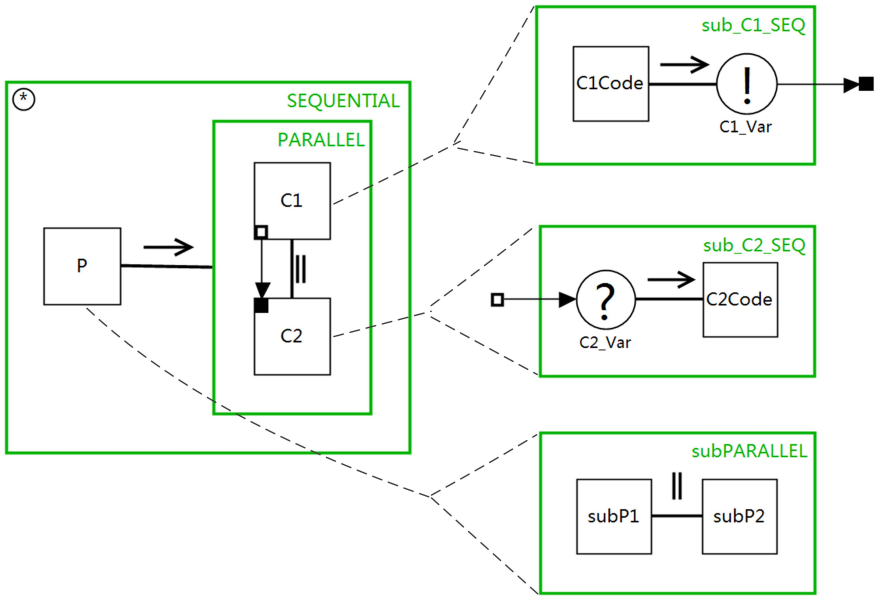

Figure 10. A TERRA Model Example.

The relations of the all model elements is visible to the users. However, the executing order of all these model elements is *not* visible. Especially, for new users (students), this is confusing, most notably when they are dealing with the Parallel composition. Adding printf() statements makes the confusion even larger, as the handling by the operating system causes displaying of these printf() statements not in the order they are generated. This motivates again the use of a simulator-like provision.

The TERRA model is transformed into a simulation model using a M2M transformation described in ETL. The transformation rules conform to both the CSP and simulation meta-models. As shown in Figure 11, the diagram containing this TERRA model is transformed into a SimDiagram with the name MainModel. Since P, C1 and C2 each have a sub-model themselves, the corresponding SimDiagram objects are also generated to describe these sub-models. In Figure 11, starting with the SimDiagram MainModel (at the left), a top-level object SEQUENTIAL is transformed from the CSP compositional group, and thus has the Start SP property (see at the top right). P contains a sub-model, thus the top-level object of SimDiagram P is transformed again from the corresponding CSP compositional group, as subPARALLEL, shown in the left column below the MainModel. Its Start SP property indicates that the elements of its parallel composition are invoked by a (pseudo) random selection. To deal with the rendezvous channel communication, if a simulation process is a writer or reader connected to a channel, then for example a writer starts first, its Next SP will be the container of the reader process. Vice versa, if a reader starts first, its Next SP will be the container of the corresponding writer, since the transformation rules know there must be a writer connected to the reader through a rendezvous channel. Thus, the rendezvous synchronization will be guaranteed, shown at the bottom right corner of Figure 11. The properties of Wr_C1 (writer) and Rd_C2 (reader) are detailed out accordingly. Note, that not all generated SimDiagrams are listed in the Figure. The properties owned by each Sim Diagram, Top-level Object or Sim Process are defined in the simulation meta-model as shown in Figure 7 and Table 1.

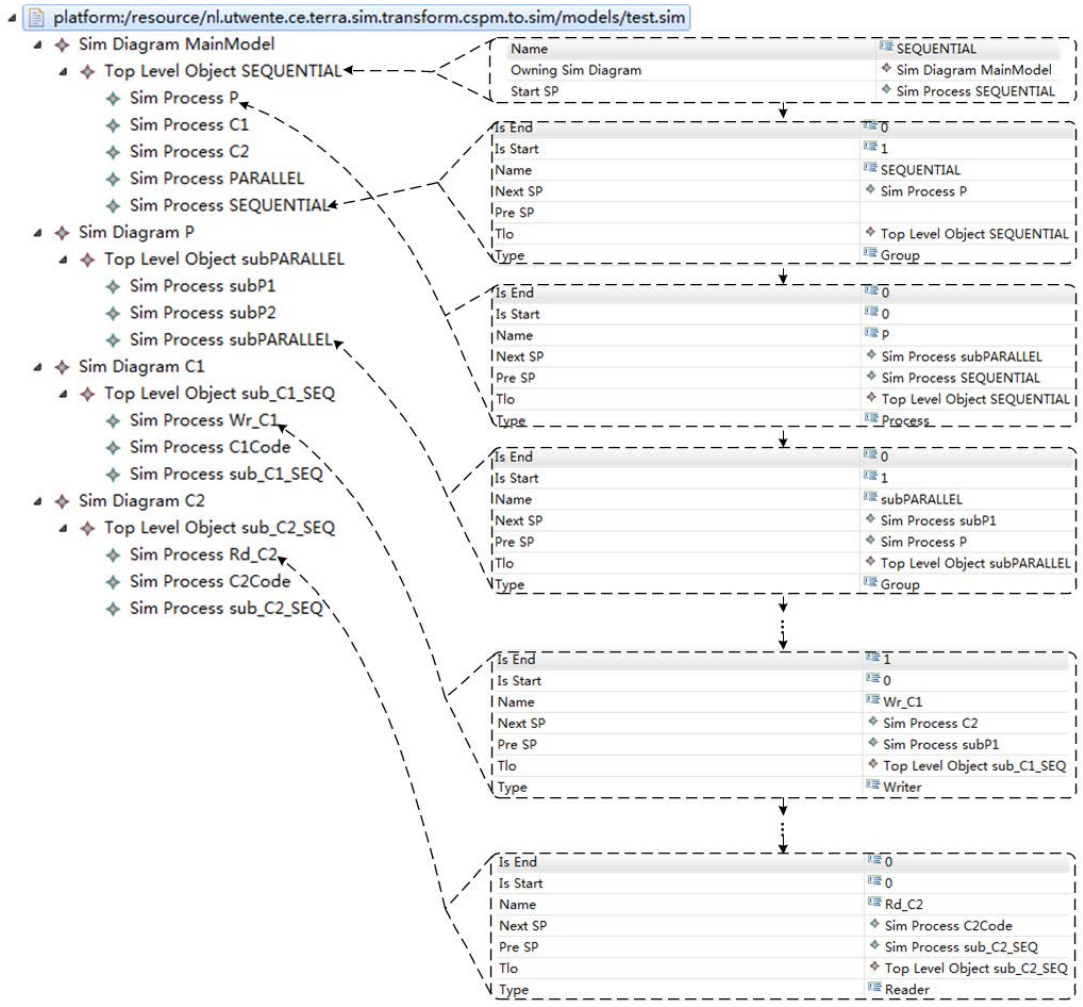

Figure 11. Partial Simulation Model in EMF Exeed Editor (built-in EMF reflective tree-based editor).

After M2M transformation from a TERRA model to a simulation model, EGL M2T

generation is invoked to take the simulation model as input, generating the simulation trace text as shown in Figure 12. From the simulation trace text, the executing order of all TERRA model elements can be observed. As mentioned before, a (pseudo) random selection is used to deal with the Parallel composition in model transformation. The simulation trace text with underline in Figure 12 shows the different executing order of subP1 and subP2 processes, coming from the same TERRA example model.

```
-- Generated by TERRA SIM to simTrace version 0.0.4
-- Input file: test.sim

-- Execution Queue
SimDiagram = MainModel
Top Level Object = SEQUENTIAL
SEQUENTIAL is Recursive
SEQUENTIAL -> Start SP = SEQUENTIAL
SEQUENTIAL -> Next SP = P

SimDiagram = P
Top Level Object = subPARALLEL
subPARALLEL -> Start SP = subPARALLEL
subPARALLEL -> Next SP = subP1
subP1 -> Next SP = subP2
subP2 = isEnd

SimDiagram = C1
Top Level Object = sub_C1_SEQ
sub_C1_SEQ -> Start SP = sub_C1_SEQ
sub_C1_SEQ -> Next SP = C1Code
C1Code -> Next SP = Wr_C1
Wr_C1 = isEnd

SimDiagram = C2
Top Level Object = sub_C2_SEQ
sub_C2_SEQ -> Start SP = sub_C2_SEQ
sub_C2_SEQ -> Next SP = Rd_C2
Rd_C2 -> Next SP = C2Code
C2Code = isEnd
```

```
-- Generated by TERRA SIM to simTrace version 0.0.4
-- Input file: test.sim

-- Execution Queue
SimDiagram = MainModel
Top Level Object = SEQUENTIAL
SEQUENTIAL is Recursive
SEQUENTIAL -> Start SP = SEQUENTIAL
SEQUENTIAL -> Next SP = P

SimDiagram = P
Top Level Object = subPARALLEL
subPARALLEL -> Start SP = subPARALLEL
subPARALLEL -> Next SP = subP2
subP2 -> Next SP = subP1
subP1 = isEnd

SimDiagram = C1
Top Level Object = sub_C1_SEQ
sub_C1_SEQ -> Start SP = sub_C1_SEQ
sub_C1_SEQ -> Next SP = C1Code
C1Code -> Next SP = Wr_C1
Wr_C1 = isEnd

SimDiagram = C2
Top Level Object = sub_C2_SEQ
sub_C2_SEQ -> Start SP = sub_C2_SEQ
sub_C2_SEQ -> Next SP = Rd_C2
Rd_C2 -> Next SP = C2Code
C2Code = isEnd
```

Figure 12. Simulation Trace Text Comparison.

4. Conclusions and future work

The main goal of this paper is to design and implement the simulation support for the TERRA tool suite. We have used model-driven design techniques, extending the existing metamodel to describe the simulation part, and using M2M transformation to generate a simulation model.

The simulation design methodology and working flow are based on the OMG MOF multi-layer specification, which provides means for the TERRA-to-simulation model transformation using different meta-models. The proposed simulation meta-model contains three levels abstracting the simulation procedure with respect to the TERRA model. Transformation rules are defined which conform to both the CSP and simulation meta-model. The hierarchical TERRA CSP model is then converted into a more flat simulation model which can be easily interpreted by the simulation engine. The M2M transformation is implemented with ETL. The natural structure of ETL and EOL brings possibilities to re-create transformation rules for other models conforming to the CPC meta-model, since most of the operations in the transformation rules are implemented in EOL. A (pseudo) random selection is implemented to deal with the Parallel CSP composition, to show the equal chance of getting executed first of all processes in the parallel composition. EGL is used to generate the simulation trace text indicating the executing order of model elements. It uses the simulation model transformed from the TERRA model as input.

The first step in future work is to show the trace text in the form of an animation of the model in TERRA, using colour coding of the status of all processes and channels. Next

plan is to extend this simulator with execution of the process contents, such that we can show the data values being transported over the channels. After that, we plan to add a co-simulation interface to let this simulator become interact-able with external models (physical world models). This will pave the road to let this simulator become a real tool for design-support of CPS software realization.

Furthermore, we have opportunities to study the effect of different execution choices for interleaving of processes, to study the implementation effects of this design freedom.

References

[1] Edward Lee et al. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369. IEEE, 2008.

[2] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. A survey of cyber-physical systems. In *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, pages 1–6. IEEE, 2011.

[3] Lui Sha, Sathish Gopalakrishnan, Xue Liu, and Qixin Wang. Cyber-physical systems: A new frontier. In *Machine Learning in Cyber Trust*, pages 3–13. Springer, 2009.

[4] Ragunathan Raj Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th Design Automation Conference*, pages 731–736. ACM, 2010.

[5] David Broman, Edward A Lee, Stavros Tripakis, and Martin Törngren. Viewpoints, formalisms, languages, and tools for cyber-physical systems. In *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, pages 49–54. ACM, 2012.

[6] Patricia Derler, Edward Lee, Alberto Sangiovanni Vincentelli, et al. Modeling cyber–physical systems. *Proceedings of the IEEE*, 100(1):13–28, 2012.

[7] Janos Sztipanovits. Composition of cyber-physical systems. In *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, 2007.

[8] The Mathworks. Simulink website. http://www.mathworks.com/products/simulink/, visited on 2015-07-31.

[9] The Modelica Association. Modelica website. https://modelica.org, visited on 2015-07-31.

[10] Controllab Products B.V. 20-sim website. http://www.20sim.com/, visited on 2015-06-01.

[11] Jan F. Broenink. Modelling, simulation and analysis with 20-sim. *Journal A*, 38(3):22–25, 1997.

[12] Ganesh Krishnamurthy. Case tools adoption and relevance. http://www.umsl.edu/˜sauterv/analysis/F08papers/View.html/, visited on 2015-06-01.

[13] Jeff C. Jensen, Danica H. Chang, Edward Lee, et al. A model-based design methodology for cyber-physical systems. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 1666–1671. IEEE, 2011.

[14] Maarten M. Bezemer, Marcel A. Groothuis, and Jan F. Broenink. Way of working for embedded control software using model-driven development techniques. In *IEEE ICRA Workshop on Software Development and Integration in Robotics, SDIR VI*, USA, May 2011. IEEE Robotics and Automation Society.

[15] Johan Eker, Jörn W Janneck, Edward Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, Yuhong Xiong, et al. Taming heterogeneity-the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[16] J. F. Broenink, Y. Ni, and M. A. Groothuis. On model-driven design of robot software using co-simulation. In *Proceedings of SIMPAR 2010 Workshops International Conference on Simulation, Modeling, and Programming for Autonomous Robots, Darmstadt, Germany*, pages 659–668. TU Darmstadt, November 2010.

[17] Jan F. Broenink, Marcel A. Groothuis, Peter M. Visser, and Maarten M Bezemer. Model-driven robot-software design using template-based target descriptions. In *ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications: How to modify and enhance commercial controllers, Anchorage, Allaska, USA*, pages 73–77. TU Braunschweig, May 2010.

[18] Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[19] Stephen D. Brookes, Charles A.R. Hoare, and Andrew W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM (JACM)*, 31(3):560–599, 1984.

[20] Maarten M. Bezemer, Robert J.W. Wilterdink, and Jan F. Broenink. Design and use of csp meta-model for embedded control software development. In *Communicating Process Architectures 2012*, volume 69, pages 185–199. Open Channel Publishing Ltd., August 2012.

[21] Herman Bruyninckx, Markus Klotzbücher, Nico Hochgeschwender, Gerhard Kraetzschmar, Luca Gherardi, and Davide Brugali. The BRICS component model: a model-based development paradigm for complex robotics software systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1758–1764. ACM, 2013.

[22] Thomas Gibson-Robinson, Philip Armstrong, Alexandre Boulgakov, and Andrew W. Roscoe. FDR3 - A Modern Refinement Checker for CSP. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *Lecture Notes in Computer Science*, pages 187–201, 2014.

[23] M. M. Bezemer, R. J. W. Wilterdink, and J. F. Broenink. LUNA: Hard Real-Time, Multi-Threaded, CSP-Capable Execution Framework. Concurrent System Engineering Series, pages 157–175, June 2011.

[24] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[25] Dan Rubel, Jaime Wren, and Eric Clayberg. *The eclipse graphical editing framework (gef)*. Addison-Wesley Professional, 2011.

[26] Dimitrios S. Kolovos, Louis M. Rose, and Richard F. Paige. The epsilon book (2010).

[27] Maarten M. Bezemer. *Cyber-physical systems software development: way of working and tool suite*. PhD thesis, University of Twente, November 2013.

[28] M. M. Bezemer and J. F. Broenink. Hardware ports - getting rid of sandboxed modelled software. In *Communicating Process Architectures 2014, Oxford, UK*, number WoTUG-36 in Concurrent System Engineering Series, pages 107–118, England, August 2014. Open Channel Publishing Ltd.

[29] Robert Shannon and James D. Johannes. Systems simulation: The art and science. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-6(10):723–724, Oct 1976.

[30] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press, 2000.

[31] OMG. Meta Object Facility (MOF). http://www.omg.org/spec/MOF/, visited on 2015-06-01.

[32] Dimitrios S. Kolovos, Richard F. Paige, Fiona A.C. Polack, and Louis M. Rose. Update transformations in the small with the epsilon wizard language. *Journal of Object Technology*, 6(9):53–69, Oct 2007. Special Issue. TOOLS EUROPE 2007.

[33] Epsilon. Documentation. http://www.eclipse.org/epsilon/doc/, visited on 2015-06-01.