Modeling and Simulation of Cyber-Physical Systems with SICYPHOS

Special Session on Evolving Design Languages and Methodologies

Frank Wawrzik, William Chipman, Javier Moreno Molina, Christoph Grimm
WG Design of Cyber-Physical Systems
University of Kaiserslautern
Kaiserslautern, Germany
{wawrzik|chipman|moreno|grimm}@cs.uni-kl.de

Abstract— **In the design of Cyber-Physical Systems, engineers from different disciplines have to co-operate, using different modeling languages and tools. In this paper we give an overview of a framework and methodology that intends to close the gap between the different disciplines: SICYPHOS. The proposed methodology is based on SysML from which model templates in different domain-specific languages are generated. Compared with state-of-the art, SICYPHOS also targets the generation of the interfaces between different domains and the integration of pre-existing modeling IP block libraries that may be written in different domain-specific languages.**[1]

Keywords—Cyber-Physical Systems; SysML; SystemC-AMS; Sicyphos;

I. Introduction

The development of Cyber-Physical Systems is a challenge that involves a number of stakeholders from different disciplines that must co-operate and communicate. Customers, managers and developers and testers with expertise in different fields, such as mechanical, electrical or software systems are all involved in the development process. To increase accuracy of communication between stakeholders, informal text-based documents are being more and more replaced with models. However, different levels require different modeling languages.

Figure 1 gives an overview of the development process. At the system level, lightweight languages like SysML [1] have become popular to describe the overall system requirements and structure. After specification and systems engineering, the components in different domains are developed using Domain-Specific Languages (DSL) such as C++ for software development, SystemC/SystemC-AMS [3] for electronic systems, and Modelica for mechanical systems. To facilitate the seamless transition from SysML to DSL, tools such as Enterprise Architect provide code generation functionality. However, this functionality is oriented to single-domain code generation, and it is insufficient to generate multi-domain models. Furthermore, it cannot benefit from reusability of pre-existing Intellectual Property (IP) block implementations.

Both different interacting domains and large IP components are essential in Cyber-Physical Systems development, and therefore, they require new approaches from research. In this paper, we give an overview of SICYPHOS (SImulation of CYber PHysical Systems). SICYPHOS is a simulation framework that integrates SysML, Modelica, SystemC, and C/C++ in a seamless way.

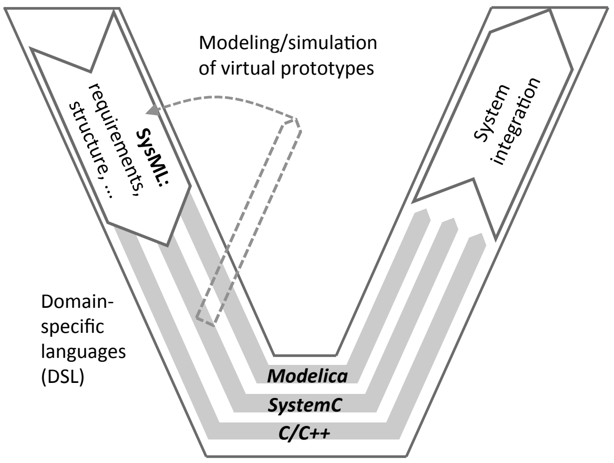

Modeling/simulation of virtual prototypes
SysML: requirements, structure, …
System integration
Domain-specific languages (DSL)
Modelica
SystemC
C/C++

Figure 1: SysML and domain-specific languages in the V diagram.

In this paper we give an overview of the SICYPHOS framework and its underlying concepts and objectives. We in particular describe concepts to achieve the intended capability of SICYPHOS to generate not only DSL, but also interfaces between different domains. These interfaces enable multi-domain model generation, as well as interfaces between IP blocks and software components to be developed. The paper is structured as follows: Section II gives an overview of state of the art, Section III presents the overall framework, Section IV illustrates a part of the concept for two practical examples and Section V concludes and gives an outlook.

[1] This work is conducted as part of the project "Integrated Components and Integrated Design of Energy Efficient Battery Systems" (IKEBA) funded by the German Federal Ministry of Education and Research (BMBF) as part of the "Energy-Efficient and Safe Electromobility" (STROM 2) measure within the program "ICT 2020 – Research for Innovation".

II. State of the Art

Complexity of Cyber-Physical System development is addressed by Model-Based Design. In Model-Based Design, virtual prototypes enable the evaluation of the system even if a physical implementation is not available. A valid virtual prototype (resp. model) reduces considerably the risks and pitfalls in system design. Models can also be reused and refined over time, improving their reliability. These characteristics become essential in Cyber-Physical System development, where a large number of devices and complex scenarios have to be validated.

However, modeling of Cyber-Physical Systems is a challenge [9]. Their heterogeneity requires the use of different, domain-specific languages that must interact concurrently in order to accurately reproduce the system behavior. However there is a lack of languages that are well-suited to model systems across fundamentally different domains.

A. Modeling and simulation

SysML provides a language to create high-level system models that may cover different domains. However, although it provides a set of diagrams and stereotypes to create high-level models, there is a lack of semantics on how to use those stereotypes, which usually results in ambiguity when translating those models into domain-specific and executable models.

Co-simulation of different domain-specific tools is possible using tools such as Ptolemy II [10]. However, the acceptance of languages and tools has been very variable depending on the modeling domain, and there are many different languages that are settled de facto standards. The adoption of new tools and languages is not always feasible. Furthermore, industry usually has comprehensive libraries of pre-existing models that have already been tested and are available for reuse. This is the motivation for new standards to reuse models regardless of their modeling language or simulation core, such as the Functional Mock-up Interface (FMI) Standard [11], which provides interfaces for model reuse and co-simulation.

The SICYPHOS framework presented in this paper proposes a SysML based top-level Cyber-Physical System modeling approach. This model is translated to domain-specific modeling languages: SystemC for hardware/software simulation, SystemC (Wireless) TLM extensions to model network and propagation, and SystemC AMS or Modelica to model analog and physical processes [12].

B. Code-generation

A basic work towards generating SystemC models from SysML models is presented in [4]. In that paper, Blocks, Flow Ports, and Operations from SysML are translated into SystemC Modules, Ports and Processes. After the model is set up, it is exported as an XMI file. A translation tool extracts the relevant data from the MOF meta-model and stores this information in C# classes. Operations are associated with state machines or existing code and after two intermediary steps (XML+XSLT) SystemC Code is generated.

A profile for detailed modeling of SystemC TLM with stereotypes is presented in [5]. Stereotypes exist for each TLM modeling construct as well as for SystemC modules and methods. The motivation of the work is to provide early consistency checking while modeling but not in the simulation. It shall reduce the overall debugging time and effort by using TLM 2.0 rules that can be statically checked as constraints. An example for an informal rule that is realized is 'Initiator cannot realize b_transport method'. Rules are formalized with OCL [2] and are constructed from the SysML meta-model. OCL is a language that enables expressing additional constraints that are difficult to express in UML. In combination with stereotypes (e.g. <<tlm_target_socket>>) the constraints are expressed. After the profile is applied to a specific implementation, a SysML tool validates the model, saves it, and transforms it to SystemC Code via XSL Stylesheets. Supported are structural and behavioral diagrams.

An approach to model SystemC-AMS with concrete semantics is described in [6]. In SystemC-AMS there are several Models of Computation (MoC), which can be coupled and used to create co-simulations of different domains and applications. In this paper each block or diagram can be assigned with a specific constraint that denotes its MoC or behavior (semantics in brackets): continuous time ('use CT'), discrete time ('use DE') and state machines ('use FSM'). Furthermore annotations of adaptor-syntax in the form of comments are assigned to ports in order to achieve a precise simulation. To achieve the efforts, a two-phase approach is applied. First the SysML meta-model is transformed into a SystemC-AMS meta-model with the Atlas Transformation Language. In a second step, the actual code is generated from the SystemC-AMS model with the Acceleo code generator.

The existing approaches introduced modeling with SysML and code generations to SystemC/TLM/AMS. With SICYPHOS we combine and extend the different approaches for generating domain-specific code from [4,5,6] and focus on generating the interfaces between different domain-specific languages, and on integrating pre-existing components.

III. SICYPHOS

SICYPHOS is a SystemC-based framework for **Si**mulation of **C**yber-**Ph**ysical **S**ystems. The SICYPHOS Framework aims to assist at demonstrating a novel design methodology for Cyber-Physical Systems. Figure **2** gives an overview of the overall framework and its design methodology. Objective is to make the transition from system-level design in Figure 1 towards the DSL, and the validation versus the requirements seamless.
To achieve this objective, the overall SICYPHOS framework, in addition to the SysML code transformations that are the focus of this work, provide the following modeling libraries and tools:

- **Virtual Prototyping Library (VPLib)**, which provides building blocks for communication (ComLib) and automotive (AutoLib) domains in the modeling language SystemC.

- **Wireless TLM** provides a wireless propagation and communication model based on SystemC TLM extensions.

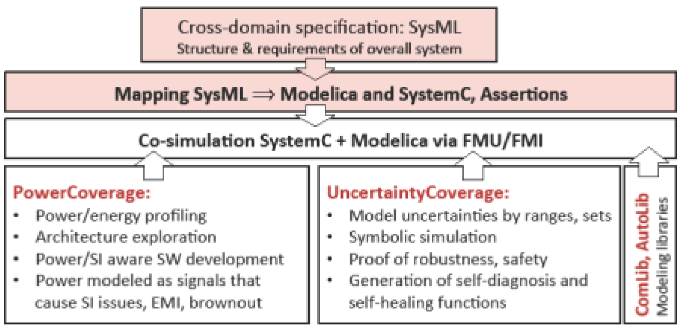

Cross-domain specification: SysML
Structure & requirements of overall system
Mapping SysML ⟹ Modelica and SystemC, Assertions
Co-simulation SystemC + Modelica via FMU/FMI
PowerCoverage:
Power/energy profiling
Architecture exploration
Power/SI aware SW development
Power modeled as signals that cause SI issues, EMI, brownout
UncertaintyCoverage:
Model uncertainties by ranges, sets
Symbolic simulation
Proof of robustness, safety
Generation of self-diagnosis and self-healing functions
ComLib, AutoLib
Modeling libraries

Figure 2: SICYPHOS Framework

- **UncertaintyCoverage** is a tool for symbolic propagation of uncertainties (e.g. aging, tolerances) in CPS.

- **PowerCoverage** is a SystemC based framework to estimate power consumption and to create meaningful high-level profiles for cross-layer energy optimization of distributed embedded systems.

A. *Design Methodology*

We assume that the overall system development starts with a cross-domain system specification using SysML. The code generators presented here then generate the modules and interfaces for all subsystems in different domains. The output is a template of a virtual prototype of the HW/SW system (C++, SystemC), including templates for the network (SystemC Wireless TLM) and propagation modeling, and templates for modeling the physical environment in SystemC AMS or Modelica.

The generation of code templates for the code to be developed in e.g. C++, but as well of models of the environment allows developers to validate designs and– code immediately against the needs of the overall system. In addition, the PowerCoverage and UncertaintyCoverage tools allow estimation of power and accuracy/robustness.

The vision is to provide a significant, yet very flexible abstraction of Cyber-Physical Systems. This abstraction shall provide the necessary means to quickly and completely set up a simulation with alternating test cases. By having a formal specification of the system, it can be set up and designed with consistent semantics.

Furthermore, developers shall be enabled to integrate pre-existing models or components without detailed knowledge. Of course functional methods and the IP have to be written by experts, but with pre-existing implementations and the appropriate ontologies, a 'non-expert' shall become capable of doing initial explorations of the architecture to get some early valuable feedback (power consumption, performance) of system behavior. If a simulation is executable in that way, the purpose is to use this information for wiser and effective design decisions in the early design phases.

B. *Implementation*

To achieve the objective, we intend to leverage a variety of methodologies in model-based and knowledge-based engineering. This paper focuses on the integration with SysML. To bridge the gap between the formal and graphical design entry and the domain specific languages we used the Acceleo code generator [7].

SICYPHOS modeling with SysML contributes in six key aspects. This work presents the demonstration of the first two, while the other outlined aspects are work in progress.

1) *IP block generation (ComLib)*

For the generation of our modules, we followed a similar approach to the code generation in [6]. We generate basic functionalities from SysML models with the Acceleo code generator. However, we did not include a model-to-model transformation and we will examine if it is beneficial for our application. Acceleo was chosen because of its pragmatism, auto completion, and because the generation was more flexible compared to code generation within Enterprise Architect. Furthermore the integration with other modeling tools and Eclipse C++ yields a complete development experience. The ComLib library is implemented in SystemC-AMS and we thus generate code for this language. The elements translations are represented in Table 1.

Table 1: SysML to SystemC-AMS Elements Translations

SysML	SystemC-AMS
Block	Module
FlowPort	Ports
Properties	Variable Declarations and Module Parameters
Instances and Connectors	Netlist
Package	Namespace
Constraints	Modeling Facilities

Additionally, we added our IP to the code generation. So far this is done sporadically from the template itself that also contains SystemC-AMS functions and code. Round-trip facilities are planned so that attributes and variable declarations can be brought back into the SysML model. Synchronization will then prevent that either model or code run out-of-date.

2) *Wireless TLM*

Wireless TLM is a framework to model wireless communication and radio propagation. For that purpose, it offers a set of SystemC modules and TLM interfaces:

- **node_base:** is the top-level API. Every top-level component with wireless connectivity must be an extension of this class.

- **wtlm_module:** is the base class for the physical layer implementation. It provides the wireless interface itself. All node_base elements must have a wtlm_module that has to be extended by the user to include application-specific aspects, such as transceiver characteristics and the modulation and bitrate defined by the protocol used.

- **network_protocol:** is the base class for any network protocol between the physical wireless interface (wtlm_module) and the top-level application (node_base).

Therefore a SysML package has been defined that includes these three basic components, as shown in Figure 4.

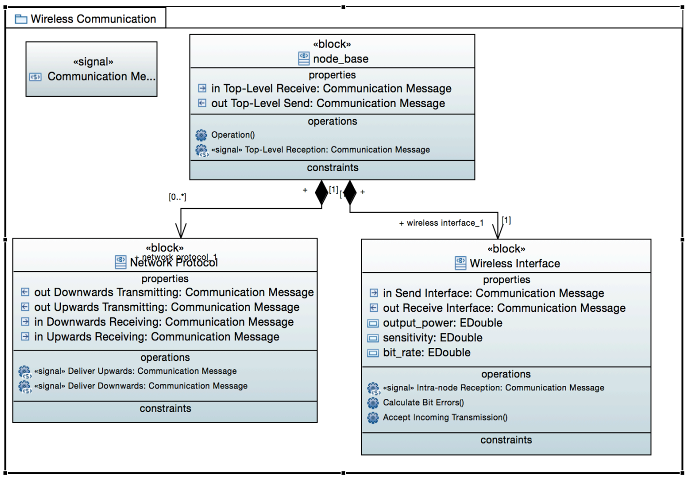

Wireless Communication
«signal»
Communication Me...
«block»
node_base
properties
in Top-Level Receive: Communication Message
out Top-Level Send: Communication Message
operations
Operation()
«signal» Top-Level Reception: Communication Message
constraints
[0..*]
+
[1]
+
+ wireless interface_1
[1]
«block»
+ network protocol_1
Network Protocol
properties
out Downwards Transmitting: Communication Message
out Upwards Transmitting: Communication Message
in Downwards Receiving: Communication Message
in Upwards Receiving: Communication Message
operations
«signal» Deliver Upwards: Communication Message
«signal» Deliver Downwards: Communication Message
constraints
«block»
Wireless Interface
properties
in Send Interface: Communication Message
out Receive Interface: Communication Message
output_power: EDouble
sensitivity: EDouble
bit_rate: EDouble
operations
«signal» Intra-node Reception: Communication Message
Calculate Bit Errors()
Accept Incoming Transmission()
constraints

Figure 4: Wireless Communication SysML Package

C. *Work in Progress*

1) Cross-domain modeling with Modelica

In the simulation of Cyber-Physical Systems we deal with a variety of systems, domains, models of computation and environments. One language is often not sufficient to cover the heterogeneity of a system. Here we decided to integrate Modelica with SystemC / SystemC AMS respectively to provide a simulation of both, the nodes, sensors as well as mechanical structures, chemical processes and environmental influences. The coupling interface development is currently being finished and will be presented in other works. It is to be generated out of connected SysML Blocks. Necessary design decisions are abstracted.

2) Converter adaptation

When we consider the semantic adaptations in [6], they really represent just the most basic. Coupling modules just within SystemC/AMS/TLM can be complex and require additional context as to what and how it is coupled. The mathematics and functionality of MoC's has to be considered. Especially, work that is done in [8] may be integrated. Amongst a variety of converters it presents a coupling of TLM with TDF by conversion.

3) Assertions

As already introduced, we use a language to specify assertions about system behavior. For instance, considering the use case with the control loop above. A simple assertion would be to check if the rising time to the reference value suffices, while, simultaneously, the settling error also stays within a certain boundary at a certain time. With Affine Arithmetic, ranges are calculated and propagated to estimate the fulfillment of the assertions.

While this work is complete in itself, experience shows that developers are hesitant to work with such rather complex expressions. We intend to abstract these assertions to provide easy to handle semantics that can be integrated in a SysML-based drag and drop design experience.

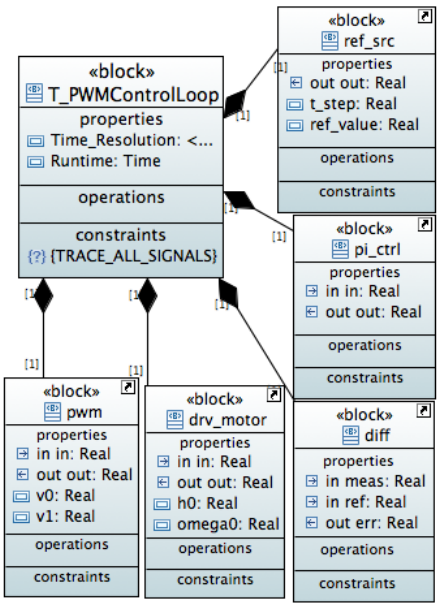


Figure 5: Block Definition Diagram of Control Loop

4) Power State Machines

SICYPHOS provides the infrastructure to estimate and track power consumption and create energy profiles that provide valuable and meaningful information across different domains. System-level power consumption estimator is implemented using finite state machines. Those state machines are the input for the power and energy profiler. State machines can therefore be specified using SysML and be used by the

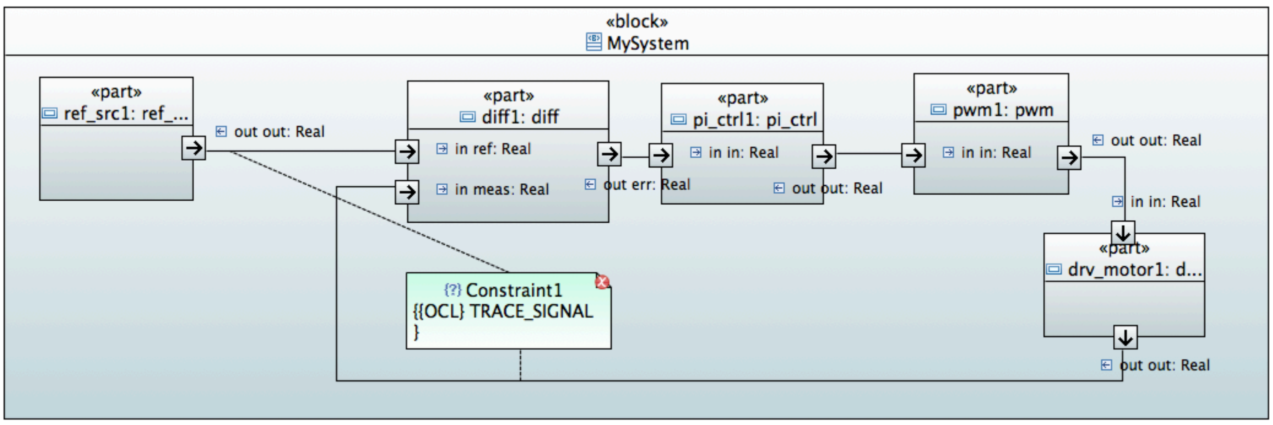


Figure 3: Internal block diagram of control loop

```
//Instantiations (unformatted)
[for (p:Package | aModel.eContents(Package))]
 [if (p.name = 'Testbenches')]
  [for (c:Class | p.eContents(Class))]
   [for (parts:Property | c.eContents(Property))]
[parts.type.name/] [parts.name/]("[parts.name/]"
[for (p:Package | aModel.eContents(Package))]
 [if (p.name = 'scp_comlib')]
  [for (cl:Class | p.eContents(Class))]
   [if parts.type.name = cl.name]
    [for (property :Property | cl.ownedElement->select(p |
                                              p.oclIsKindOf(Property)
and not p.oclIsKindOf(Port)))], [property.name/]
    [/for]
   [/if]
  [/for]
 [/if]
[/for]);
```

Figure 6: Acceleo mapping for module instantiations

energy profilers to provide complex and abstract energy consumption estimations of software, communication, and even distributed tasks. Furthermore, state machines can also be used to assess the consistency of power consumption and to identify possible risks, such as brownout or crosstalk.

IV. USE CASES

A. *PWM Control Loop*

As a demonstration for the modules in our 'comlib' library, we chose to model a simple control loop with SysML in Papyrus. The loop consists of a pwm, a PI controller, a difference module and a load. It regulates to a reference current source.

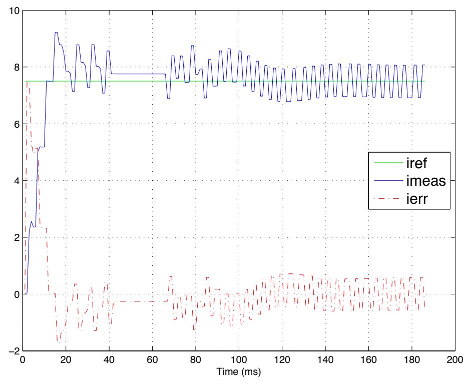


Figure 7: Simulation results

The Block definition diagram of this example is shown in Figure 5.

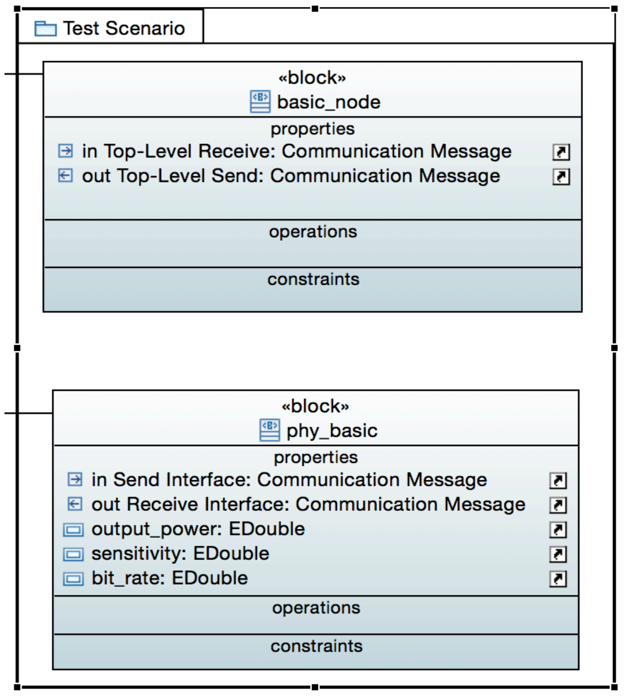
The block 'T_PWMControlLoop' is in this case the system and testbench, which contains the control loop as subblocks. Parameters to be defined at this level are the 'Time_Resolution', which determines the step width and thus accuracy of the simulation. 'Runtime' defines the total runtime of the simulation. The framework will also offer a variety of modeling facilities. The tracing of signals is a simple example that is used by SystemC-AMS itself and which we implemented as a constraint on the testbench with 'TRACE_ALL_SIGNALS'. It ensures tracing of all signals in the testbench as well as storing trace data to an appropriate log file. Ports are defined as type real with their corresponding direction.

Through the reference source block, the simulation timestep 't_step' and the actual reference current value 'ref_value' are adjusted. For the PWM just the high and low plateau values of the modulation are shown. It also offers the changing of periodic time / frequency and ramp time slope values. The load 'drv_motor' represents the degree of an opening of a throttle valve and is modeled via a resistance and inductance.

The implementation of the use case is shown in Figure 3 within an internal block diagram. Parts are instances of their definition from the 'blocks' of the block definition diagram. Their ports are connected via SysML Connectors. In this example single signals can also be traced with the 'TRACE_SIGNAL' constraint. The constraint is allocated to a signal with a constraint link.

To bridge the gap between the formal definition in SysML and an executable simulation we used the Acceleo code generator as mentioned above. It is easily integrated into the design flow as a java project, but can also be a stand-alone application. Acceleo provides a mapping language that enables accessing the SysML and UML meta-models with their respective properties as well as OCL elements and constraints. A simple programming syntax yields the most important functionality. In case it is required, queries or Java Services can be written additionally. The mapping has been implemented for this test case. Figure 6 shows the mapping of the parts in the test case of Figure 3 without the connections. Basically it is iterated through the packages, classes and properties in the UML model file and corresponding variables are accessed. OCL is used to filter the properties.

Afterwards we run the simulation and obtain a



Test Scenario
«block»
basic_node
properties
in Top-Level Receive: Communication Message
out Top-Level Send: Communication Message
operations
constraints
«block»
phy_basic
properties
in Send Interface: Communication Message
out Receive Interface: Communication Message
output_power: EDouble
sensitivity: EDouble
bit_rate: EDouble
operations
constraints

Figure 8: SysML Package of the Test Scenario

characteristic PI control behavior. Figure 7 shows this with traces of the reference value, the control error, as well as the measured current.

B. *Wireless Scenario*

To setup a wireless scenario, the user must extend the Wireless TLM base classes with application specific methods and parameters. Figure 8 shows a SysML package that contains specializations of the SysML blocks provided in the Wireless Communication package presented in Section III.B.2).

This way, the user can define the specific physical layer with the specific parameters and methods of the transceiver and protocol being used. The user can also define the network protocols.

Finally with the node_base specialization the user can implement the application and define the communication stack architecture. However, in order to define the communication stack, a SysML Internal Block Diagram (IBD) must be used, where all protocols are defined as parts of the top-level node block and the flow-ports are connected in the appropriate order. Figure 9 shows the IBD of a ZigBee node implementation. This way, the internal blocks of the node can be instantiated and the TLM sockets can be bound based on the SysML flow-port connectors.

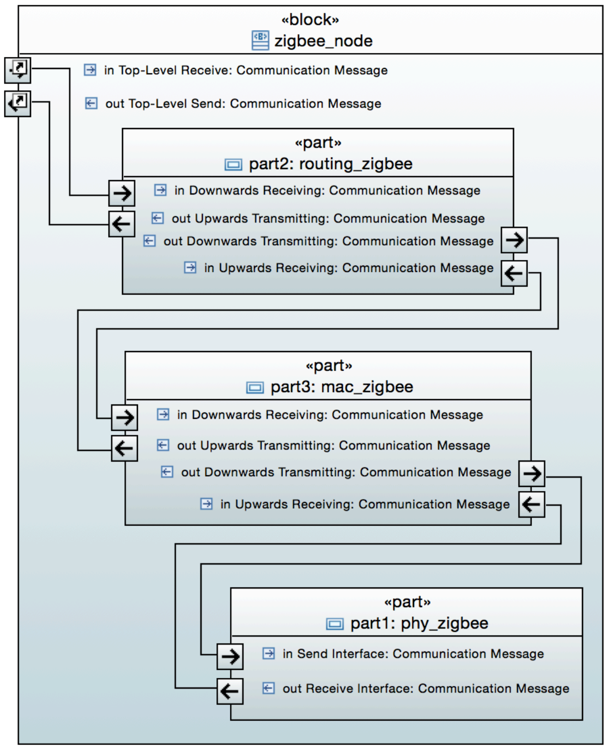

«block»
zigbee_node
in Top-Level Receive: Communication Message
out Top-Level Send: Communication Message
«part»
part2: routing_zigbee
in Downwards Receiving: Communication Message
out Upwards Transmitting: Communication Message
out Downwards Transmitting: Communication Message
in Upwards Receiving: Communication Message
«part»
part3: mac_zigbee
in Downwards Receiving: Communication Message
out Upwards Transmitting: Communication Message
out Downwards Transmitting: Communication Message
in Upwards Receiving: Communication Message
«part»
part1: phy_zigbee
in Send Interface: Communication Message
out Receive Interface: Communication Message

Figure 9: Internal Block Diagram of a ZigBee node

V. Conclusion

We have given an overview of the SICYPHOS framework. We proposed a SysML based approach to create system-level cross-domain specifications of CPS that can then be translated into different domain-specific languages and their interfaces to concurrently simulate all different parts.

While major parts of the framework are operational, the integration of our framework into SysML is work in progress. This paper presented the code generation of SystemC-AMS models based on the ComLib, a communication building block library, and the code generation for the Wireless TLM library, that provides wireless communication interfaces to model both the communication protocol stack and the radio propagation. Both libraries are provided by SICYPHOS.

This way, using SysML syntax and diagrams the user can generate complex scenarios to validate CPS including the physical environment and wireless networks. The user can then for instance rapidly and visually define and simulate networks of nodes with different communication stack architectures, enabling exploration of network protocols. As part of the future work, we intend to provide generation of interfaces with more modeling languages, such as Modelica, which has valuable libraries for modeling physical processes. We plan also to track cross-domain requirements such as power and energy consumption and enable verification of cross-domain issues.

References

[1] OMG, "Systems Modeling Language (SysML) specification," OMG standards, formal/2013-06-01

[2] OMG, "Object Contraint Language (OCL) specification," OMG standards, formal/2014-02-03

[3] Accelera, "SystemC AMS 2.0 Reference Manual", 2013. [online]. Available: accellera.org

[4] Brasil, Keyla; da Silva Junior, Diogenes; "Automatic translation of SysML models to SystemC exectuable specification," *Workshop on Circuits and System Design*, pp. 1–4, 2011. [Online]. Available: http://www.lbd.dcc.ufmg.br/colecoes/wcas/2011/0021.pdf

[5] Jain, Vaibhav; Kumar, Anshul; Panda, Preeti, "A SysML Profile for Development and Early Validation of TLM 2.0 Models," In 7th Proc. European Conf. Modelling Found. Applicat. (ECMFA'11), volume 6698 of LNCS, pages 299–311. Springer, 2011.

[6] Cafe, Daniel Chaves; dos Santos, Filipe Vinci; Hardebolle, Cecile; Jacquet, Christophe; Boulanger, Frederic, "Multi-paradigm semantics for simulating SysML models using SystemC-AMS," *Specification & Design Languages (FDL), 2013 Forum on* , vol., no., pp.1,8, 24-26 Sept. 2013

[7] J. Musset, E. Juliot, S. Lacrampe, W. Piers, C. Brun, L. Goubet, Y. Lussaud, and F. Allilaire, "ACCELEO user guide," 2006. [Online]. Available: acceleo.org

[8] Damm, Markus; "Crossing Modeling Paradigms in System Models", Ph.D. Thesis, TU Wien, March 2015.

[9] Derler, Patricia, Edward A. Lee, and A. Sangiovanni Vincentelli. "Modeling cyber–physical systems." *Proceedings of the IEEE* 100.1 (2012): 13-28.

[10] Brooks, Christopher X. ; Lee, Edward A. ; Tripakis, Stavros: Exploring models of computation with ptolemy II. Proc. of the 8th IEEE/ACM/IFIP international conf. on Hardware/software codesign and system synthesis. New York, NY, USA : ACM, 2010 (CODES/ISSS '10).

[11] Modelica Association Project: Functional Mock-up Interface for Model Exchange and Co-Simulation. In: Functional Mock-up Interface Standard (2013)

[12] Molina, J. M., Damm, M., Haase, J., Holleis, E., & Grimm, C. (2014). Model based design of distributed embedded cyber physical systems. In *Models, Methods, and Tools for Complex Chip Design* (pp. 127-143). Springer International Publishing.