An Approach Combining Simulation and Verification for SysML using SystemC and Uppaal

Abbas Abdulhameed
Université de Franche-Comté
FEMTO-ST/DISC UFR ST
abbas.abdulhameed@femtost.fr

Ahmed Hammad
Université de Franche-Comté
FEMTO-ST/DISC UFR ST
ahammad@femto-st.fr

Hassan Mountassir
Université de Franche-Comté
FEMTO-ST/DISC UFR ST
hmountas@femto-st.fr

Bruno Tatibouet
Université de Franche-Comté
FEMTO-ST/DISC UFR ST
btatibou@femto-st.fr

ABSTRACT

Ensuring the correction of heterogeneous and complex systems is an essential stage in the process of engineering systems. In this paper we propose a methodology to verify and validate complex systems specified with SysML language using a combination of the two techniques of simulation and verification. We translate SysML specifications into SystemC models to validate the designed systems by simulation, then we propose to verify the derived SystemC models by using the Uppaal model checker. A case study is presented to demonstrate the effectiveness of our approach.

Keywords

SysML, SystemC, Simulation, Model checking, Acceleo, ATL, Uppaal.

1. INTRODUCTION

Over the last decade, the complexity of systems has considerably grown since these systems integrate an increasing number of components and a variety of technologies. At the same time, system engineers always have to reach the following main objectives; building the right systems, correctly and on time, while reducing costs.

The correct design of complex systems continues to challenge engineers. Bugs in a design that are not uncovered in early design stages can be extremely expensive. Simulation is a predominantly used techniques supported by tool to validate a design in the industry. Formal verification overcomes the weakness of exhaustive simulation by applying mathematical methodologies to validate a design. The work described here focuses upon a technique that integrates the best characteristics of both simulation and formal verification methods to provide an effective design, building them correctly and on time, while reducing costs. One purpose of modeling is to enable the analyst to predict the effect of changes to the system. Models may describe requirements, the behavior and/or the structure of the designed system, it may be used to validate the characteristics of some part or of the whole of the designed system, e.g. Its functionality or its performance. Thus, the model-based systems engineering approach using SysML[24] can be used to specify graphically all aspects of complex systems consisting of components from heterogeneous domains, in particular hardware and software. SysML is a modeling language that permits to easily obtain a specification of a complex system including structural and behavioural aspects.

An important issue in modeling is model validity. Model validation techniques include simulating the model under known input conditions and comparing model output with system output. Generally, a model intended for a simulation study is a mathematical model developed with the help of simulation software. SysML lacks formality for the required validation. Translating SysML-based specification into SystemC [9] environments allows to enable rigorous static, and dynamic system analysis. The joint use of languages SysML and SystemC is a good way sees to satisfy the requirements of the specification and simulation. SystemC is a language with C++ - like syntax that enables the description of concurrent systems in an event-based way. It is able to describe systems from the executable specification level.

Interestingly the question of how simulation and verification can be combined to validate the characteristics of some part or of the whole of the designed system, e.g. Its functionality or its performance. We think that performance requirements can be validated by simulation, but validation of system functionalities requires the use of the formal methods of check. To combine SysML and SystemC is not enough for validating these systems, Sometimes, it is necessary to validate functional requirements by using techniques of the formal methods checking, yet SysML and SystemC do not supply tools for the check. The integration of the model checker Uppaal[5] in this approach would allow to complete the process of validation. Uppaal is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata.

The main objective of this paper is to describe a methodol-

ogy of modelling, simulation and validation of complex systems. This one uses the techniques of MDE(Model Driven Engineering) for the transformation of models (SysML and SystemC). We use the works presented in [3] for the extraction of Uppaal timed automata from SystemC. The second section of this paper addresses the state of the art of the combination of language (eg SysML) and a code environment simulation same as SystemC. The third section, describing the example of controls of traffic lights, is used to illustrate our approach and serves as an experience feed-back. The process of validation as well as thus the proposed approach are presented in the section 4, two requirements(functional and performance) are taken into account to validate them. Finally, in Section 5, we conclude and we outline some ideas for future works.

2. RELATED WORKS

We present works related to combining SysML with SystemC for validating and verifying systems. In [8] and [29], the authors defined a design methodology and a development flow for the hardware, based on a UML4SystemC profile and encompassing different levels of abstraction. Both SystemC and C profiles are consistent groups of modelling constructs designed to lift the programming features include structural and behavioral of the two coding languages to the UML modeling level. In [19], the paper describes a SystemC profile which is a consistent set of modeling constructs designed to lift both structural and behavioral attributes, of the SystemC language to SysML level. It provides means for software and hardware engineers to improve the current industrial(SoC), design methodology joining the capabilities of SysML and SystemC to operate at system-level by include events and time attributes. In [27], the integration is based on a mapping from the SysML to the SystemC for the structural and behavioral aspects. The refined co-design flowing starts from a SysML description at a high abstraction level. Then, it proceeds through a series of refined SystemC models, to lower abstraction levels of design. The more complex last-level SystemC coding is left to automation. An implementation allows to efficiently simulate the behaviors of the system. Before synthesizing hardware description, the verification of SystemC designs is important, to reveal errors in early stages of the development flow. In [9], an approach was developed to introduce SysML as UML profile for the co-modeling of SystemC and C with code generation support. It was implemented in the context of Artisan Studio [18]. Many implementations that consider the transformation of SysML designs into SystemC code generation, such as the Altova UModel [28], that designs application models and generate code and project documentation, then refines designs and completes the round trip by regenerating code, that makes visual software design practical for any project. The Enterprise Architect software [23]. Supports advanced MDA transformations using easy to edit transform templates, with generation and reverse engineering of source code for SystemC language, this can quickly develop detailed solutions from abstract models.

Formal verification of SystemC has recently gained significant interests [16]. Compared to simulations, formal verification provides full coverage, and then may improve the reliability and the robustness of the design. In[30],[15] the formal verification of SystemC designs was including assertion-based dynamic validation, symbolic simulation, explicit-state model checking, and symbolic model checking. There are more than software tools are presented that encapsulate the scope of SystemC design to formal verification approaches, such as the KRATOS [12],[13] verifies safety properties, in the form of program assertions, by allowing users to explore two directions in the verification. First, by relying on the translation from SystemC designs to sequential C programs, is capable of model checking the resulting C programs using the symbolic lazy predicate abstraction technique. Second, are implements a novel algorithms, called ESST (Explicit-Scheduler/Symbolic-Threads) and S3ST that combines explicit state techniques to deal with the SystemC scheduler, with symbolic techniques to deal with the threads, which are is built on top of NUSMV and MATHSAT, and uses state-of-the-art SMT-based techniques for program abstractions and refinements. In [11],[21] the authors proposed Promela encoding of SystemC to analysis, the kinds of property and search behaviors during the verification. The technique is effective and efficient in finding bugs, describe the encoding of the SystemC scheduler into an asynchronous formalism, the input language for *("SPIN")*. In [20] IVL *("Intermediate Verification Language")* combine and adapt POR (Partial Order Reduction) and symbolic execution techniques to allow the verification and the simulation of SystemC designs. IVL presents some limitations. It does not implement loop detection in the proposed symbolic simulator. In our paper, a methodology that combines SysML with SystemC and Uppaal is proposed to validate requirements of complex system. A first translation from SysML to SystemC is done to validate non functional properties by simulation. Then, a second translation from SystemC models to Uppaal is achieved to provide a formal verification of functional properties.

3. BACKGROUND

3.1 SysML

SysML is a language of modelling specified by the OMG. It is a language of graphic modelling with semi-formal semantics, availability scopes are to improve UML-based [17], complex systems development processes with the successful experiences from the system engineering discipline.

3.1.1 BDD and IBD

The Block Definition Diagram *"BDD"*, is specified by its *parts*, and its *flow ports*. The physical components of the block is referred to *Parts* and the interfaces of block is referred to *Flow ports* [27]. Internal Block Diagram *"IBD"* includes properties so that its values, parts, and references to other blocks can be specified. However, created for a block (as an inner element) will only display the inner elements of a classifier (parts, ports, and connectors).

3.1.2 SMD

The State Machine Diagram *"SMD"* facilitates the specification of a state-based behaviour of any component whose behaviour can be thus expressed; some components will additionally require other mechanisms to define their nonstate-based behaviour [31]. The SMD defines a set of concepts that can be used for modeling discrete behaviour through finite state transition systems. The state machine represents behaviour as the state history of an object in terms of its transitions and states.

A block includes operations so that its values, parts, and references execution blocks can be specified. However, whereas an SMD created from a block will display the inner elements of a classifier, through the *transition, entry, and exit*, the states are specified along with the associated event and guard conditions. The state can be active or inactive, state has incoming transitions and outgoing transition; when an outgoing transition is fired, the state is exited and becomes inactive. When an incoming transition is fired, the state is entered and it becomes active. Activities that are invoked while in the state are specified as*do Activities*, and can be either continuous or discrete. A composite state has nested states that can be sequential or concurrent[24].

3.2 SystemC

SystemC[2], is an open-source system-level design language based on C++ that has its own simulation kernel. The SystemC is a run-time scheduler that handles both the synchronization and scheduling of concurrent processes. The designers can apply object-oriented capabilities to hardware design. SystemC allows to work at a higher level of abstraction, enabling extremely faster, more productive architectural trade-off analysis, design functional level modeling describes modeling done at levels above Transaction Level Modeling (TLM) and encompasses System Architectural Models(SAM)with System Performance Models[10].

Modeling at this level is algorithmic in nature and may be timed or untimed. Models may represent software, hardware or both the typical behavior is described as generators and consumers of data. Processes may be assigned time for performance analysis purposes, this timing does not cycle accurate but rather describes the time to generate or consume data or to model buffering or data access. The behavior of the interfaces between modules is described using communication protocols. These types of models are used to explore architectures, for proof of algorithms and for performance modeling and analysis. SystemC processes execute concurrently and may suspend on *"wait()"* statements. Such processes require their own independent execution stack which called *"SC_THREADs"*, when the only signal triggering a process is the clock signal "clk" we obtain what we call *"SC_CTHREADs"* (clocked thread process) certain processes do not actually require an independent execution stack and cannot suspended on *"wait()"* statement, such processes are termed *"SC_METHODs"*. Processes are executing in zero simulation time and returns control back to the simulation kernel.

Simulation Semantics of SystemC

A high-level executable design allows to efficiently simulate the behaviours of the complex system before synthesizing the RTL hardware description. The verification of SystemC designs is of fundamental importance, to detect errors in early stages of the development flow, and prevent expensive propagation down to the hardware.

Formal verification of SystemC is lately winning significant interests. Compared to simulations, formal verification provides full coverage, and thus may enhance the reliability and the robustness of the design.

The controller of SystemC design is scheduled when execution. It controls the simulation time, the execution of processes, manages event notifications and updates primitive channels. SystemC supports the notion of delta-cycles, the delta cycle is not clock cycle and no time having advanced. Delta-cycles are used to simulate new updates and event triggered processes to be simulated from the current execution phase of current time. A brief simulation steps are as follows:

1. Initialization phase: each process is executed once,
2. Evaluation phase: execute all schedule processes in current run queue.
3. Update phase: update value and add new triggered runnable processes to waiting queue or queue of t + n.
4. If the queue (t+0) is not empty, move queue (t+0) to run queue and go to step 2.
5. If waiting queue (t+0) is empty, advance time to closest time step, and move queue (t+x) to run queue and go to step 2.
6. If queue is empty, it means no event needs to be simulated, then simulation ends.

Which is using delta notification, the event and its triggered processes are scheduled to be run immediately after current execution and update phase.

3.3 Uppaal Timed Automata

Timed Automata[7] is a Finite-State Machine extended with clock variables. The concept of model time-dependent behavior of clock variables, which are used as clock constraints. Systems comprising multiple concurrent processes are modeled by networks of timed automata, which are executed with interleaving semantics and synchronize on channels. Uppaal [5],[6] is a toolset for the modeling, simulation, animation and verification of networks of timed automata. The Uppaal model checker enables the verification of temporal properties. The simulator can be used to visualize feed-back produced by the model checker.

Uppaal Model Checking Tool

The Uppaal modeling language extends timed automata by introducing parameterized timed automata templates, bounded integer variables, binary and broadcast channels, and urgent and committed location. Timed automata templates provide the possibility to model similar timed automata only once and to instantiate them arbitrary often with different parameters.

A Uppaal model comprises three parts: *"global declarations"*, *"parameterized timed automata"* and a *"system declaration"*. In the global declarations section, global variables, constants, channels and clocks are declared. The timed automata templates describe timed automata that can be instantiated with different parameters to model similar process. In the system declaration, the templates are instantiated and the system to be composed is given as a list of timed automata. To enable the implementation of formal verification techniques, the dynamic requirements are formalized into TCTL

properties *"Time Computation Tree Logic"* [25], TCTL is an extension of CTL *"Computation Tree Logic"* which allows considering several possible future from a state of system. The model checker Uppaal has four TCTL quantifiers [14]

3.4 Metamodels

MDE recommends the use of models at different levels of abstraction. The model is an abstract view of reality and conforms to a metamodel that precisely defines the concepts present at this level of abstraction and the relationships between the concepts, therefore the metamodel allows of representing complex mechanisms involving multiple concepts, a written report in a given metamodel will be said according to the metamodel. The metamodeling approach means that "a metamodel is used to specify the model that comprises (SysML, SystemC)".

Model transformation

Model transformation represents the heart which becomes an MDE predominant activity in the development process. To the principle of model transformations has attracted much attention by becoming a subject of research for the academy and industry. The transformation term models remain quite broad as a consequence of studies have been done order to define categories and criteria for model transformation by allowing developers to choose an approach as needed.

Several conditions were adopted order to define the transformation process that generally describes by the conversion of a certain level of model abstraction complies to a metamodel to a target at a certain level of abstraction compliance with its metamodel whose passage is described by of the rules of transformation,

4. OVERALL APPROACH

This section introduces the set of activities related to the specification and the verification approach used to assess the correction of complex systems designed with SysML, simulated by SystemC, and verified using Uppaal. Figure 1 summarises the main steps of the proposed approach. First, we have to create the SysML diagrams *"BDD, IBD, SMD"*, for specifying the system structure and behaviour. Then, the SysML diagrams are mapped into SystemC modules. Then, the derived SystemC specification is mapped into Uppaal automata. Properties to verify by uppaal are derived from SysML Requirements Diagram *"RD"*.

5. THE MAPPING TECHNIQUE

5.1 Mapping SysML to SystemC

In this section, we will focus on how to implement (and what are the parameters used to affect the implementation), and learn the structure and behaviour modeling and generate a SystemC specification. The mapping methodology is used to create SystemC code from SysML diagrams will focus over the next subsections.

5.1.1 Design SysML Diagrams

The SysML diagrams are modeled using the TOPCASED tool. Topcased is a graphical tool that capture SysML diagrams, with our combination the SysML and SystemC profiles start from a system description given as the input conditions of

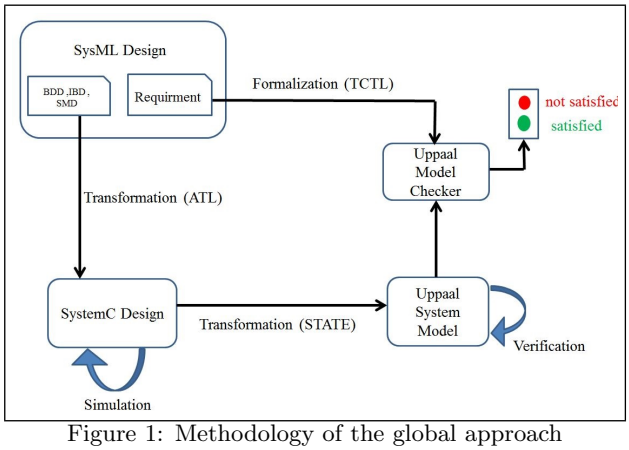

Figure 1: Methodology of the global approach

1. *structural view* by a SysML BDD and an IBD of the top-level block used to encapsulate the overall hierarchical design. In addition, the IBDs for the design of each compound block with the associated BDDs for the block types definition. The basic mapping between SysML and SystemC is
   A. SysML Blocks → SystemC Modules
   B. SysML Flow Ports → SystemC ports

2. *behavioral view* by a SysML SMD of the overall system functionality associated with the top-level block to model input, output, sequences, and conditions, for coordinating the inner blocks behaviors.
   A.SysML Operations → SystemC Processes

5.1.2 From SMD to SystemC

The aim of extending the model with state machine diagram is to introduce and translate behavioral information about the system to the target language, in this case SystemC. There are different possible translations of the considered semantics with the behavioral concepts of SystemC[4]. The following translation rules were chosen

1. A SMD is mapped into SystemC process.
2. A SystemC *Threads* are used to allow parallel states activation semantics.
3. The activation of each state is represented by a boolean signal, more than one state can be active at the same time.
4. A SystemC *Threads* are sensitive each one to another by notification to represent every event trigger which can possibly fire transition from the associated state.
5. In SystemC *Threads*, condition statements are used to represent the guards used in SMD.

By applying this rules, the basic mappings of SMD elements into SystemC Processes are as follows:
A. State → State of the process(case statement)

B. Peudostate → State of the process(condition statement)
C. Transition → Action of the process(event statement)
D. Do activity → Action of the process(action statement)

5.1.3 *Description of the implementation approach*

The general process of our approach consists of several stages, in the first place the modeling with SysML diagrams which will be the source models for the transformation. Our purpose in this work is the transformation of three diagrams: BDD, IBD and SMD diagrams. With model transformation "Model2Model", was chosen ATL language, such as language and the transformation method allowing passing a SysML model to a model SystemC. The application of the methodology with ATL is based primarily on

1. The definition of the source and target metamodel.
2. The definition of the style of transformation.
3. The definition of the source model that conforms to source metamodel.

The source metamode represent the SysML metamodel and the metamodel target will SystemC. Both are carried out under the metamodel formalizes of Eclipse EMF Ecore, the different stages of implementation are shown in Figure 2.

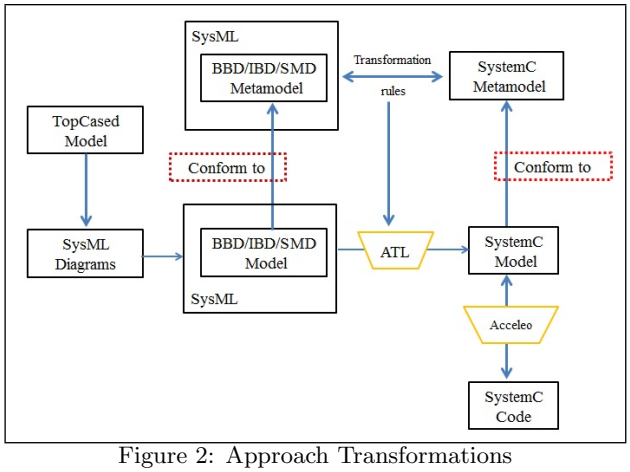

Figure 2: Approach Transformations

5.1.4 *Transformation with ATL*

After the definition the metamodel of SysML, SystemC and models sources of SysML diagrams, we use ATL as transformation language models. With the aim of achieving the previously defined rules ATL declarative *"rule"* is used. ATL rule is characterized by two mandatory elements:

1. A pattern on the source model *"from"* with a possible constraint.
2. One or more grounds of the target model *"to"* that explain how target elements are initialized from the corresponding source element.

When creating a target item from a source element, ATL retains a traceability link between the two elements. This link is used to initialize a target item in the *"to"* match as seen in Listing 1.

The ATL following code shows an example of the rules used in the ATL model transformation

Listing 1: rule model to model

```
rule Model2SCModel{
        from sysml: MMUML!Model(
                sysml.oclIsTypeOf(MMUML!Model)
        )
        to scModel: MMSystemC!SCModel(
                name <- sysml.name
        )
}
rule Package_BDD2SystemC_Main {
        from
        BDD :MMUML!Package(
                BDD.oclIsTypeOf(MMUML!Package)
                )
        to
                Top : MMSystemC!SC_object (
                name <- BDD.name,
                ownerScModel <- BDD.getModel()
                )
}
```

5.1.5 *Code Generation*

Acceleo is a language code generator which allows generating structured file from an Eclipse Modeling Framework(EMF) [22] model, the output is a text that can be a programming language or other formalism. Acceleo requires defining an EMF metamodel and a model conforming to metamodel that will result into text.

Once this definition is done, then we can execute the code generator, in this example, have the metamodel and model of SystemC for code generation, we need to create an Acceleo project and configure the workflow necessary to code generation specifying the link between the generator, the metamodel and model.

After it is needed to define the Template code using the keywords of SystemC language and attributing information from the SystemC model of transformation. In the first line of Acceleo code we are importing the metamodel so that the generator knows the structure of our model. The important concept to define Acceleo is also called Template, it is the smallest unit identified in a template file, and allows to define the main reference for the workflow order to collect information from the necessary to model code generation. Figure 3,4 illustrates the SysML2SystemC code.

5.2 Mapping SytemC to Uppaal

To provide a formal verification of SystemC designs, we use existing tools to transform, by sequence of refinement steps, a systemC design into an Uppaal automata. Our objective is to analyse a SystemC design with model-checking techniques. As shown in Figure 5, the tool STATE (SystemC to Timed Automata Transformation Engine)[26] is used to translate an abstract SystemC design into a Uppaal model and the Uppaal model-checker is used to check properties

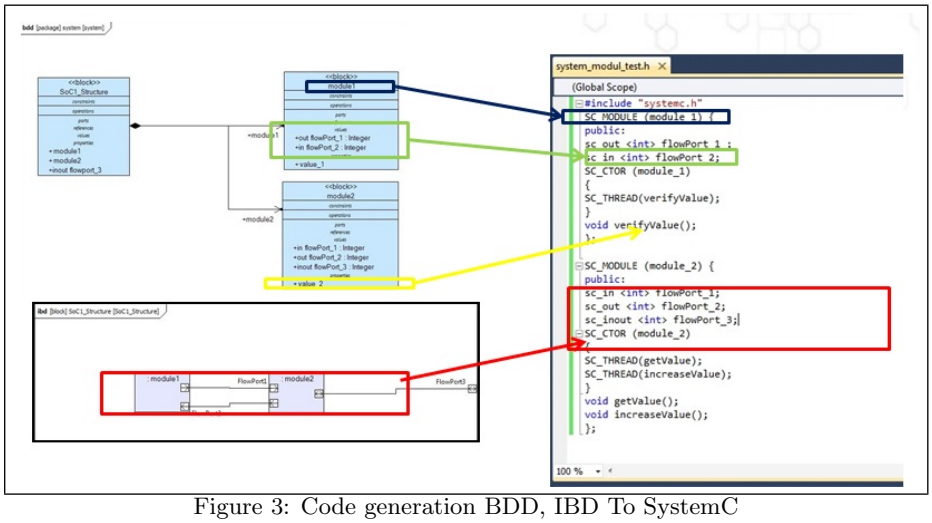

Figure 3: Code generation BDD, IBD To SystemC

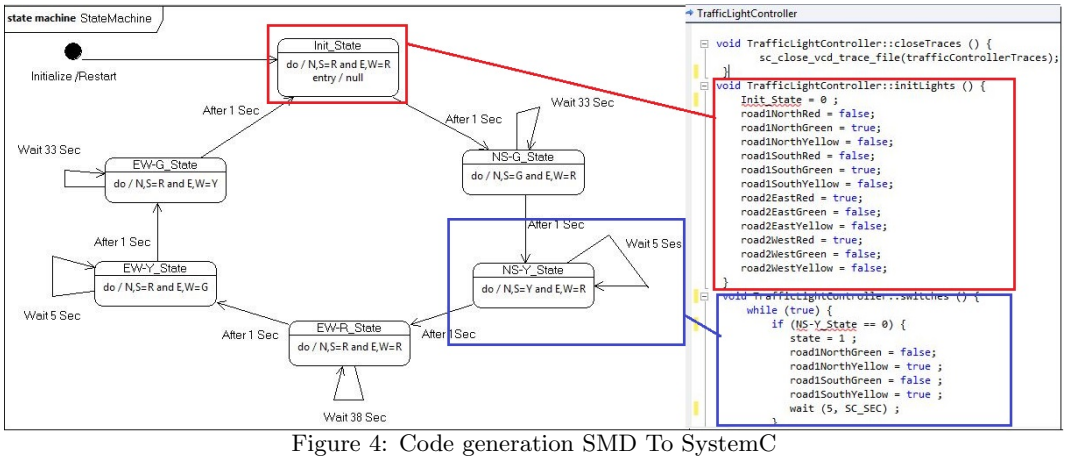

Figure 4: Code generation SMD To SystemC

expressed as temporal logic formulas. Verification results express satisfaction or no satisfaction of properties. If a property is not satisfied, the Uppaal model checker additionally generates a feed-back, which can be used for debugging purposes. The feed-back can also be visualized and animated in the Uppaal tool to understand where the problem appear from.

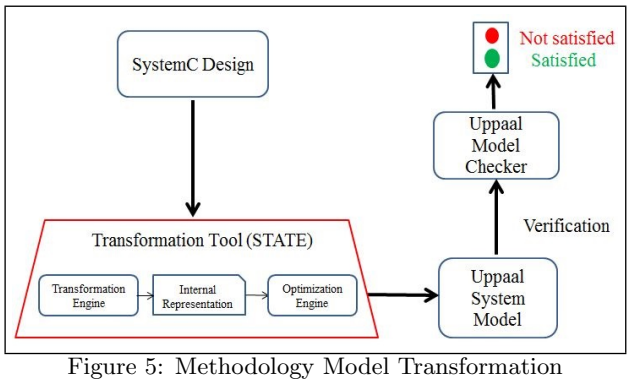

Figure 5: Methodology Model Transformation

6. CASE STUDY

This section discusses a case study with the aim to present the effectiveness of our approach to specify, validate and verify the behavior of road intersection signals. The state sequence of the car flow in the crossroad is the base of the traffic behavior. Some additional features have been added to make this workbench complex enough to measure meaningful evaluations of development system properties. They are managed by a system that synchronizes the color changes of the different junction lights. The traffic-light colors are managed by a controller which depends on the number of cars waiting to cross the junction. The methodology and code generator presented were used, for example, to show it, BDD with six blocks. The first block is the most abstract level of the modeling Crossroads block named CrossRoad represents the system as a whole, it is composed of three sub-blocks("Controller System, NorthandSouthLights, EastandWestLights")and sub-sub blocks ("Timer, Road Sensor , Camera"). Figure 6 illustrates the crossroads top level modeling.

To represent the internal structure of the Crossroads block by IBD. The diagram shows the flow ports, the port management allow continuous moving the direction of Controller System and the port of other parts (i.e. "NorthandSouthLights, EastandWestLights"). Figure 7 shows the IBD di-

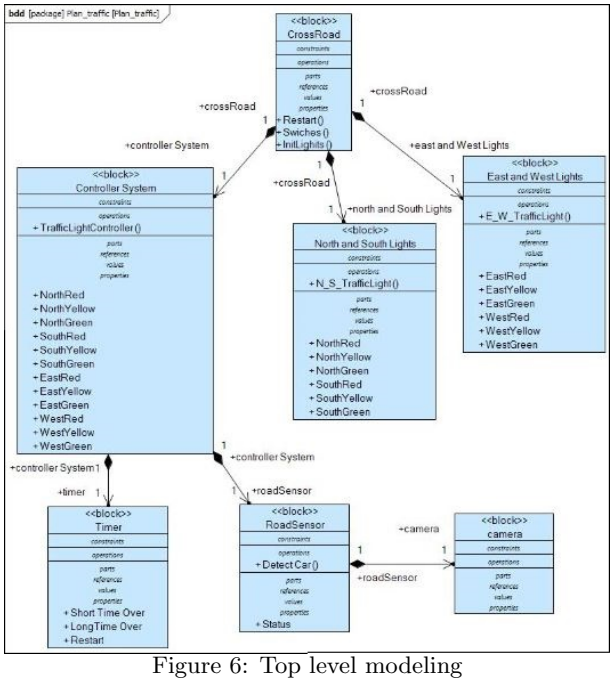

Figure 6: Top level modeling

agram. To represent the behavior of the*"Controller System"* block by SMD. The diagram shows the state of operation*"TrafficLightController"*. Default state is made the next state if no transition line condition is satisfied as shown in state *"Initialize"*. If the state has any unconditional Transition Line, then assigning default state to next state is omitted as shown in case *"State0"*, we see that the next state conditions appear in the generated SystemC code according to the assigned priority. Consider the following situation where,*"NS"* and*"WE"* are two inputs for a state machine.
"if (NS==1) and (WE==0) NextState = State1"
"else if (NS==1) NextState = STate2"
If both *"NS and EW are 0"*, then NextState is dependent on the order of the appearance of the conditions in the code. Figure 8 shows the SMD diagram. The timer for the crossfires is managed by a controller system which at each start up, initializes a clock that measures the duration for each crossfire color: red (38 Sec.), yellow (5 Sec.), and green (33 Sec.), then it sends the value to NorthandSouthLights and EastandWestLights, based on the given entries **"NorthRed, North Yellow,...WestGreen"**. Figure 10 show the results of validation of the Crossroades system.

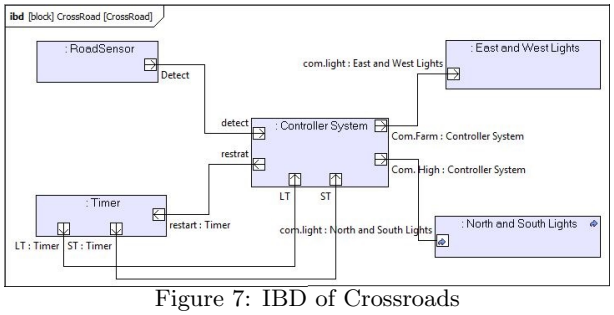

Figure 7: IBD of Crossroads

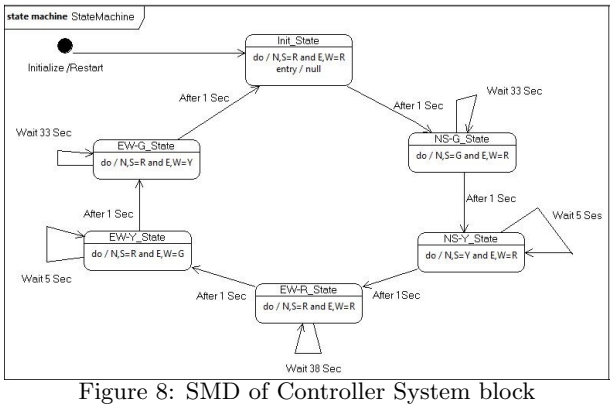

Figure 8: SMD of Controller System block

6.1 Simulation

When SystemC code is successfully generated from the SysML representation of the Crossroades system (Figure 6, 7 and 8), the subsequent step is to simulate the generated SystemC design. In Figure 9, we show the simulation results. The simulator shows the state of each light as true and false values through the time. There we can verify that no green light on North and South lights is turned on when there is also a green light on the East and West lights.

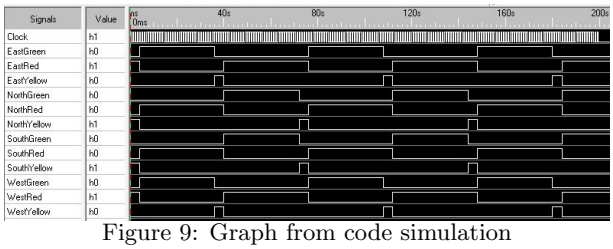

Figure 9: Graph from code simulation

6.2 Verification

To verify requirements of the Crossroades system, the SystemC design is translated into Uppaal automata. Requirements are expressed as temporal logic properties expressed in TCTL. First, we verify that the system is deadlock free. we express this property in TCTL by the formula: **"AG not deadlock"** . Then, we verify time properties. As example, we verify that both NorthandSouth Lights and EastandWest Lights can not stay in yellow color more than 5 second, which is expressed in TCTL as:
A. AG(NorthSouth.NY imply NorthSouth.cn<5).
B. AG(EastWest.SY imply EastWest.cs<5)).
Then, we verify that both NorthandSouth Lights and EastandWest Lights can not stay in red color more than 38, which we express in TCTL as:
A. EG(NorthSouth.cn>38).
B. EG(EastWest.cs>38).
In Figure 10, we present Uppaal environment, where Uppaal automata, properties and their verification results are shown.

7. CONCLUSION AND FUTURE WORK

In this paper we proposed a methodology for verifying and validating complex systems designed with SysML. We have

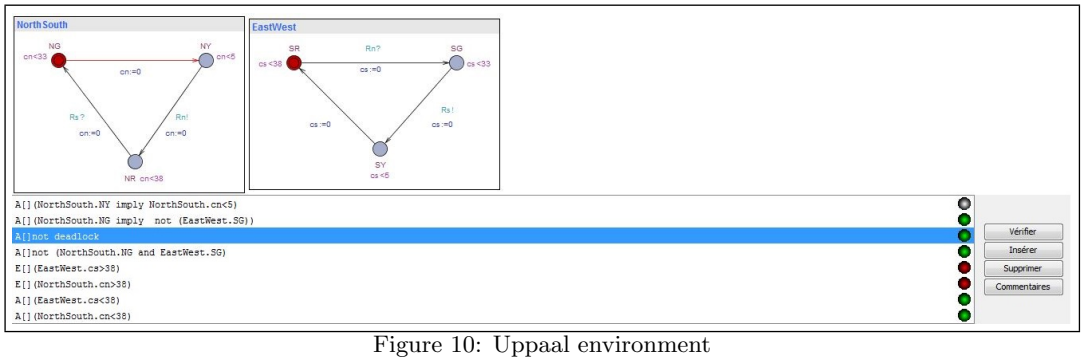

Figure 10: Uppaal environment

proposed a model transformation from BBD, IBD and SMD SysML diagrams to SystemC. SystemC code is generated as text files and can be used for simulation to validate the designed systems. Then, based on STATE tools, the derived SystemC specifications were translated into Uppaal automata for verifying SysMl requirements by using Uppaal model checking. We illustrated the practicability of our approach by a case study where the transformation from BBD, IBD and SMD SysML diagrams to SystemC was implemented on Topcased platform using ATL and Acceleo tools and the transformation form SystemC to Uppaal automata was achieved by STATE tools. Obtained results of experimentations and simulations are encouraging. In future, we plan to investigate SystemC code generation from other SysML diagrams like Activity and Sequence behaviour diagrams allowing the translation of more aspects of a system.

8. REFERENCES

[1] Optimized Transformation and Verification of SystemC Methods], author=Pockrandt, Marcel and Herber, Paula and Gross, Holger and Glesner, Sabine, booktitle=Pre-Proceedings of the 12th International Workshop on Automated Verification of Critical Systems (AVoCS 2012), volume=2, pages=1561, year=2011.

[2] IEEE Standard for Standard SystemC Language Reference Manual. *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pages 1–638, 2012.

[3] A. Abdulhameed, A. Hammad, H. Mountassir, and B. Tatibouët. An Approach based on SysML and SystemC to Simulate Complex Systems. In *MODELSWARD 2014, 2nd Int. Conf. on Model-Driven Engineering and Software Development*, pages 555–560, Lisbon, Portugal, Jan. 2014. Short paper. 6 pages.

[4] G. Agosta, F. Bruschi, and D. Sciuto. UML Tailoring for SystemC and ISA Modelling. In *UML for SOC Design*, pages 147–173. Springer, 2005.

[5] G. Behramm, A. David, and K. G. Larsen. A tutorial on UPPAAL. *proceedings of the 4th International School on Formal Methods for the Design of Computer*, 2004.

[6] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on*, pages 125–126. IEEE, 2006.

[7] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, pages 87–124. Springer, 2004.

[8] D. C. Black. *SystemC: From the ground up*, volume 71. Springer, 2010.

[9] M. Bombino and P. Scandurra. A model-driven co-simulation environment for heterogeneous systems. *International Journal on Software Tools for Technology Transfer*, pages 1–12, 2012.

[10] F. Boutekkouk. Automatic SystemC code generation from UML models at early stages of systems on chip design. *International Journal of Computer Applications*, 8(6):10–17, 2010.

[11] D. Campana, A. Cimatti, I. Narasamdya, and M. Roveri. An analytic evaluation of SystemC encodings in promela. In *Model Checking Software*, pages 90–107. Springer, 2011.

[12] A. Cimatti, A. Griggio, A. Micheli, I. Narasamdya, and M. Roveri. KRATOS–A Software Model Checker for SystemC. In *Computer Aided Verification*, pages 310–316. Springer, 2011.

[13] A. Cimatti, I. Narasamdya, and M. Roveri. Software model checking SystemC. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(5):774–787, 2013.

[14] A. Friesen. On Challenges in Enterprise Systems Management and Engineering for the Networked Enterprise of the Future. In *Enterprise Interoperability*, pages 1–2. Springer, 2011.

[15] D. Groβe, U. Kühne, and R. Drechsler. HW/SW co-verification of embedded systems using bounded model checking. In *Proceedings of the 16th ACM Great Lakes symposium on VLSI*, pages 43–48. ACM, 2006.

[16] R. Hajisheykhi, A. Ebnenasir, and S. Kulkarni.

[17] M. Hause, A. Stuart, D. Richards, and J. Holt. Testing safety critical systems with SysML/UML. In *Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on*, pages 325–330. IEEE, 2010.

[18] M. C. Hause, C. Gloucestershire, and G. Hn. ARTiSAN Software Tools. 2008.

[19] V. Jain, A. Kumar, and P. Panda. Exploiting UML based validation for compliance checking of TLM 2 based models. *Design Automation for Embedded Systems*, 16(2):93–113, 2012.

[20] H. M. Le, D. Große, V. Herdt, and R. Drechsler. Verifying SystemC using an intermediate verification

language and symbolic simulation. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–6. IEEE, 2013.

[21] K. Marquet, B. Jeannet, and M. Moy. Efficient encoding of systemc/tlm in promela—full version. Technical report, Technical Report TR-2010-7, Verimag Research Report, 2010.

[22] G. Nicolescu, I. O'Connor, and C. Piguet. *Design technology for heterogeneous embedded systems*. Springer Publishing Company, Incorporated, 2011.

[23] O. Nikiforova, N. Pavlova, K. Gusarovs, O. Gorbiks, J. Vorotilovs, A. Zaharovs, D. Umanovskis, J. Sejans, et al. Development of the Tool for Transformation of the Two-Hemisphere Model to the UML Class Diagram: Technical Solutions and Lessons Learned. In *Proceedings of the 5th International Scientific Conference „Applied Information and Communication Technology*, pages 11–19, 2012.

[24] OMG. OMG Systems Modeling Language (OMG SysML^{TM}) Version 1.3. 2012.

[25] W. Penczek, B. Woźna, and A. Zbrzezny. Towards bounded model checking for the universal fragment of TCTL. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 265–288. Springer, 2002.

[26] M. Pockrandt, P. Herber, H. Gross, and S. Glesner. Optimized Transformation and Verification of SystemC Methods. 2012.

[27] E. Riccobene and P. Scandurra. Integrating the SysML and the SystemC-UML profiles in a model-driven embedded system design flow. *Design Automation for Embedded Systems*, pages 1–39, 2012.

[28] B. Scholtz, A. Calitz, and I. Snyman. The usability of collaborative tools: application to business process modelling. In *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, pages 347–358. ACM, 2013.

[29] Y. Vanderperren, W. Mueller, D. He, F. Mischkalla, and W. Dehaene. Extending UML for Electronic Systems Design: A Code Generation Perspective. In *Design Technology for Heterogeneous Embedded Systems*, pages 13–39. Springer, 2012.

[30] M. Y. Vardi. Formal techniques for systemc verification; position paper. In *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*, pages 188–192. IEEE, 2007.

[31] S. K. Wood, D. H. Akehurst, O. Uzenkov, W. G. J. Howells, and K. D. McDonald-Maier. A model-driven development approach to mapping UML state diagrams to synthesizable VHDL. *Computers, IEEE Transactions on*, 57(10):1357–1371, 2008.