Using UML for Domain-Specific Component Models[1]

Ana Petričić, Luka Lednicki
Faculty of Electrical Engineering and Computing
University of Zagreb, Croatia
{ana.petricic, luka lednicki}@fer.hr

Ivica Crnković
Mälardalen University, Sweden
Mälardalen Research and Technology Centre
{ivica.crnkovic}@mdh.se

ABSTRACT

Over the recent years there is a tendency for using domain-specific languages which enable expressing design solutions in the idiom and level of abstraction appropriate for a specific problem domain. While this approach enables an efficient and accurate design, it suffers from problems of standardization, portability and transformation between the models. This paper addresses a challenge of tailoring UML, a widely used modelling language, for domain-specific modelling. We discuss a possible solution for achieving interoperability between UML and the domain-specific language SaveComp Component Model (SaveCCM) intended for real-time embedded systems, by means of implementing a transformation between UML and SaveCCM models. The challenge of the transformation is to keep all necessary information including the domain specific semantics. The paper presents the strategy for the transformation, its implementation and an analysis. We also address the second challenge, a usability of the domain-specific language (i.e. SaveCCM) in comparison with usability of extended UML and by an experiment analyse its usability in comparison with SaveCCM.

1. INTRODUCTION

A number of Domain Specific Languages (DSL) exists nowadays which provide more expressiveness at the design time and efficiency in analysis and testing. One of such DSLs is the SaveComp Component Model (SaveCCM) [1][2], intended for building embedded control applications in vehicular systems. SaveCCM is a research component model in which design flexibility is limited to facilitate analysis of real-time characteristics and dependability. As a domain specific language, SaveCCM is productive for designing safety-critical systems responsible for controlling the vehicle dynamics.

A disadvantage of DSL is paradoxically, its specificity – it may require additional efforts to be used, it can cause obstacles in communication of design decisions between different stakeholders, and it requires development of custom design tools. Contrary to most DSLs, the Unified Modeling Language (UML) [3] as a de facto standard in industry has a wide spread base of trained users and a number of modelling tools. By combining UML and SaveCCM, we could take advantages of both languages in different aspects they provide, and in different stages of system development process. Using of UML for domain specific modelling can reduce time and cost of building specific modelling tools, and can bring the feature of portability and standardization to a system model. However there can be a challenge to a) to express a DSL by UML and b) implement a transformation between them.

In this paper we set up two questions. The first one is the feasibility of combining general-purpose and domain specific languages in terms of full and unique transformation of models in both directions. The second question is the usability of our approach, compared to using only standard, domain specific modelling in order to perceive if there is any need for building specialized tools instead of using general purpose ones with appropriate extensions. These two questions we apply on SaveCCM and UML. The specificity of the case is the domain, namely real-time and embedded systems, which requires quite different modelling, due to specific interaction styles and specific concerns, such as real-time properties and resource constraints. To obtain the answer to the first question we provide a solution for achieving interoperability between SaveCCM and UML through a formal way of representing SaveCCM models using UML 2.0 component diagram and defining a transformation between the two model formats. Usability is discussed over the results of an empirical evaluation.

The rest of the paper is organized as follows: After an overview of our approach in section 2, we bring a brief description of SaveCCM in section 3. In sections 4 and 5 we present our UML profile and the design of transformation between UML and SaveCCM. Section 6 discusses applicability of our approach and presents results of an empirical evaluation that we have conducted. Finally, section 7 presents related work and our concluding marks are given in section 8.

[1] This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS, and the UKF supported by Croatian Government and the World Bank via the DICES project, and EU FP7 Q-Impress project.

2. THE SAVECCM OVERVIEW

SaveCCM is a component model intended for designing safety-critical resource constrained systems responsible for controlling the vehicle dynamics. SaveCCM technology provides a support for designing systems and analysis of their timing properties built in an integrated development environment named SaveIDE. The main architectural elements in SaveCCM are: (i) *Components*, which are the basic units of encapsulated behaviour with a functionality that is usually implemented by a single function written in C programming language. Besides the C function, each component is defined by associated ports and optionally quality attributes. (ii) *Switches*, which provide facilities to dynamically change the component interconnection structure (at configuration or run-time); thus allowing a conditional transfer of data or triggering between components. (iii) *Assemblies*, which provide means to form aggregate components from sets of interconnected components and switches. SaveCCM also provides a hierarchical component composition mechanism in a form of a special type of a component – *composite component*, where the functionality of a component is specified by an internal composition instead of using a C function.

An important characteristic of SaveCCM is the distinction between data transfer and control flow, which is achieved by distinguishing two kinds of ports; *data ports,* where data of a given type can be written and read, and *trigger ports* that control the activation of components. The separation of data and control flow allows a model to support both periodic and event-driven activities. In addition to ports, the interface of a component can contain quality attributes, having each attribute associated with a value or a model-based specification and possibly a confidence measure. These attributes can hold the information about the worst case execution time, reliability estimates, safety models, etc.

An example of a simple temperature regulation system modelled using SaveCCM (in SaveIDE tool) is shown on Figure 1. It consists of two SaveCCM components, one assembly and one composite component. On the figure are also visible various types of SaveCCM ports: input and output trigger ports (triangle shape), input and output data ports (square shape), and input and output combined ports (combined triangle and square shape).

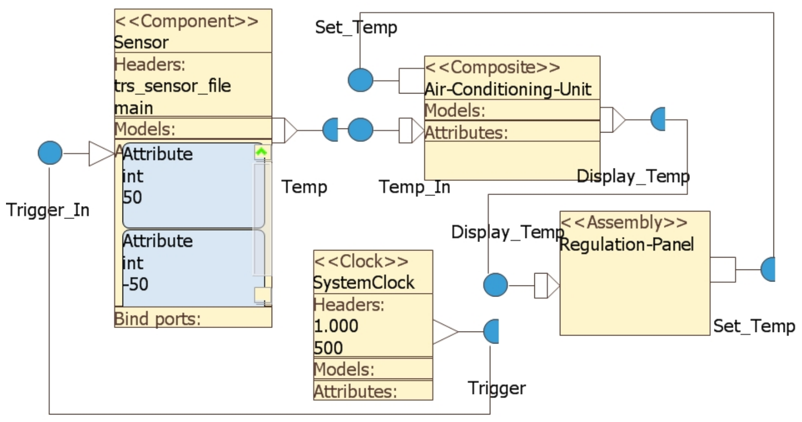

Figure 1. SaveCCM model of Temperature regulation system

More information on SaveCCM with a detail description of model elements and their attributes, as well as an overview of the SaveCCM execution model can be found in SaveCCM reference manual [1] and [2].

Apart from unique modelling elements, such as switches or assemblies that provide specific behaviour, as well as clock and delay components, SaveCCM introduces several valuable concepts that cannot be found in UML.

- The distinction between data transfer and control flow.
- Concept of component interface. In SaveCCM the functional interface of every modelling element is defined by a set of ports associated to the element and optionally, quality attributes.
- Model analysis and verification. SaveCCM uses quality attributes for defining non-functional properties of components and systems which allow analysis of various properties and system verification.
- Execution semantics of active model elements, defined to provide run-time model analysability. The execution model is rather restrictive, its basis is the pipes and filter control-flow paradigm in which component execution is defined by a sequence of activities: start by trigger, read, execute, and write.

3. THE SAVEUML PROFILE, A UML SPECIFICATION OF SAVECCM

A common way of specialising UML to align it with important design issues in different domains is to define a UML profile suitable for the domain. We will use the UML profiling mechanisms to tailor UML for SaveCCM domain in a controlled way. By defining a profile we generate an extension to UML consisting of elements with different semantics. However we also must limit the use of standard UML elements to a subset that fits our target domain. Our UML profile, named *SaveUML profile*, defines a one-to-one mapping to elements defined by SaveCCM. We have identified the UML 2.0 subset that addresses the concepts used in component-based development as well as the ones existing in SaveCCM. It includes the UML 2.0 *Components* and *Composite Structures* packages. We call this subset a *UML 2.0 component model.*

The UML profile we developed, named *SaveUML profile*, is to provide an equivalent language to SaveCCM language. It aims at modelling systems in UML but using SaveCCM semantics, and supporting unambiguous transformation between the UML and SaveCCM models

Considering that UML profiles are a standard UML extension mechanism and are therefore a part of UML, they are as widely recognized as UML itself and should be supported by all standard modelling CASE (Computer-Aided Software Engineering) tools. This possibility of customizing UML for specific domain purposes while remaining within boundaries of the UML standard and keeping the possibility of using UML CASE tools, presents a reasonable motivation for customizing and using UML instead of a specific modelling language.

The process of defining the SaveUML profile consisted of three phases:

1. *Identification of SaveCCM and UML component model elements*. We have made a detail analysis of UML 2.0 component model (a subset of UML concerning UML components), which allowed us to survey the similarities between UML and SaveCCM and identify compatible elements. In addition we defined mapping rules for all SaveCCM elements that need to be translated to UML elements and corresponding UML elements that can be used for mapping.
2. *Identification of SaveCCM language constraints*. Designing SaveCCM elements with UML 2.0 elements brought up various problems resulting from a strict syntax of SaveCCM and the universality of UML. Therefore, we had to create a set of constraints to refine the UML 2.0 component model semantics to be suitable for designing SaveCCM models.
3. *Translation of previously identified elements,* during which a suitable UML element is found for every SaveCCM language elements. Chosen UML elements were then further customized through the use of necessary stereotypes, properties and constraints.

The diagram of the SaveUML profile that developed in is depicted in Figure 2. The SaveUML profile specifies a set of stereotypes which extend elements of the UML 2.0, namely UML `Component`, `Port`, `Property`, `Artifact`, `Usage` and `Dependency`. Each element from SaveCCM domain has its corresponding element in the SaveUML profile. For introducing the properties of SaveCCM elements (e.g. jitter and period attributes of SaveCCM clock component etc.) we used the tagged value mechanism. The SaveCCM semantics is imposed upon the UML model using Object Constraint Language (OCL).

During the process of creating the SaveUML profile, we have made several design decisions considering representing of SaveCCM architectural elements within the profile, the method of defining substructure of components and different concepts of interfaces in SaveCCM and UML: (i) *Components.* Since SaveCCM introduces three main architectural elements (component, assembly and switch.) and three subtypes (clock, delay and composite component) we had to define six new UML elements by using stereotypes that will extend the UML `Component` element. Similar concept was applied for defining different port types that exist in SaveCCM language. (ii) *Subcomponent*s. SaveCCM offers two elements that may have an internal structure: assembly and composite component. In UML 2.0, we can specify internal subelement of a component either as its property (by using the `Property` metaclass) or as a packaged element (using the `PackageableElement` metaclass). We chose the latter approach – using `PackageableElement`. Definition of an owning component also includes the definition of its subcomponents, leaving no need for referencing outside elements. Such a definition of subcomponents is also referred as an *embedded definition of components*. (iii) *Interfaces.* In SaveCCM the functional interface of every modelling element is defined by a set of ports associated with the element. Because of semantic differences of interface in the SaveCCM and UML, we decided not to use UML interfaces in SaveUML profile.

Using In order enforce the SaveCCM semantics to the SaveUML profile we defined a number of constraints within the profile using the Object Constraint Language (OCL) [3]. We used OCL constraints to enforce the SaveCCM semantics and restrict the usage of UML concepts that do not have equivalent elements within SaveCCM. We divided the implemented constraints into two main groups – *Restrictions on UML* and *SaveCCM semantics*. Each group has several sub-groups which are described in Table 1. In total we implemented 117 constraints. We found that identifying and specifying OCL constraints is the major part in development of a UML profile.

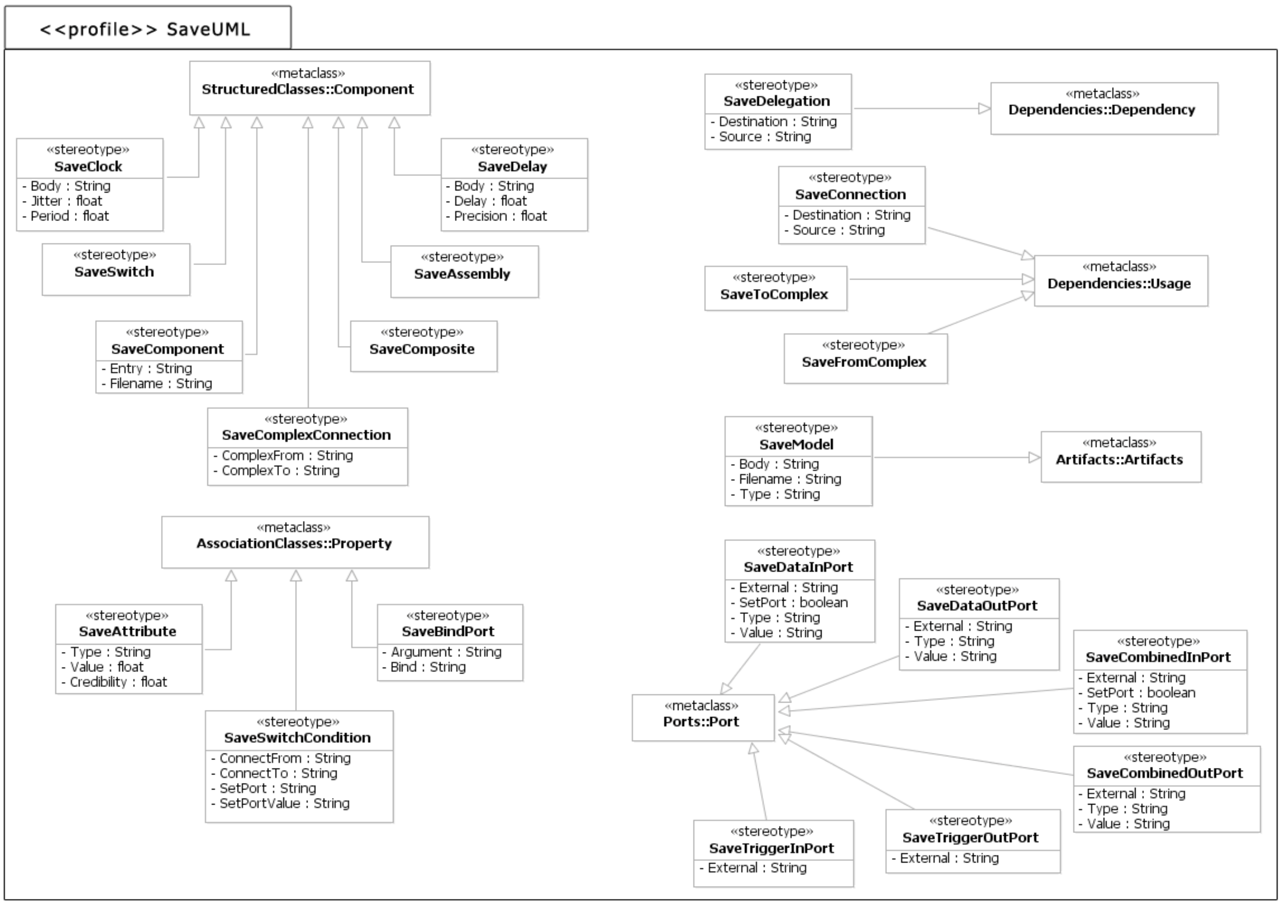
<<profile>> SaveUML
«metaclass» StructuredClasses::Component
«stereotype» SaveClock - Body : String - Jitter : float - Period : float
«stereotype» SaveDelay - Body : String - Delay : float - Precision : float
«stereotype» SaveSwitch
«stereotype» SaveAssembly
«stereotype» SaveComponent - Entry : String - Filename : String
«stereotype» SaveComposite
«stereotype» SaveComplexConnection - ComplexFrom : String - ComplexTo : String
«metaclass» AssociationClasses::Property
«stereotype» SaveAttribute - Type : String - Value : float - Credibility : float
«stereotype» SaveBindPort - Argument : String - Bind : String
«stereotype» SaveSwitchCondition - ConnectFrom : String - ConnectTo : String - SetPort : String - SetPortValue : String
«stereotype» SaveDelegation - Destination : String - Source : String
«metaclass» Dependencies::Dependency
«stereotype» SaveConnection - Destination : String - Source : String
«stereotype» SaveToComplex
«metaclass» Dependencies::Usage
«stereotype» SaveFromComplex
«stereotype» SaveModel - Body : String - Filename : String - Type : String
«metaclass» Artifacts::Artifacts
«stereotype» SaveDataInPort - External : String - SetPort : boolean - Type : String - Value : String
«stereotype» SaveDataOutPort - External : String - Type : String - Value : String
«stereotype» SaveCombinedInPort - External : String - SetPort : boolean - Type : String - Value : String
«metaclass» Ports::Port
«stereotype» SaveCombinedOutPort - External : String - Type : String - Value : String
«stereotype» SaveTriggerInPort - External : String
«stereotype» SaveTriggerOutPort - External : String

Figure 2. Diagram of the SaveUML profile

Table 1. Constraints implemented in SaveUML profile

Constraint group	**Count**
Restrictions on UML	**56**
Forbidden connections. Restrictions on UML 2.0 considering using various types of connectors. Also, connectors should not connect elements directly etc.	**17**
Using interfaces. Using UML interfaces is not allowed within the SaveUML profile; these constraints deal with this issue.	**12**
Substructure definition. Internal structure of an element may only be defined using packaged elements. Further, the only allowed packaged element is a `Component`	**6**
Number of stereotypes. Even though UML has the option to apply multiple stereotypes to one element, in SaveUML profile, one element can have only one stereotype applied	**21**
SaveCCM semantics	**61**
These constraints are defining attributes that main SaveCCM elements may own.	**6**
Owning ports. Since SaveCCM offers several kinds of ports, each port must have appropriate stereotype applied in order to determine its type. Further, some SaveCCM elements have restrictions on number of ports that they own	**13**
Bind port. These constraints introduce semantic rules considering special type of port – *bind port.*	**3**
External ports. These constraints introduce semantic rules considering special type of port – *external port.*	**6**
Switch semantics. Switch component is specific SaveCCM element. These constraints introduce its semantics. They deal with concept of *set port*, *switch condition* and *switch connection*.	**5**
Connections between SaveCCM elements. Since SaveCCM offers two kinds of connections, each connector must have an appropriate stereotype applied. Also, depending on the connection type, cyclic connections are forbidden or allowed. Finally, these constraints ensure conformance of the connected ports (their types and directions).	**23**

4. SAVEUML TRANSFORMATIONS

The transformation approach is based on using the eXtensible Stylesheet Language for Transformations (XSLT) [4]. Recommended by the World Wide Web Consortium (W3C), XSLT is a flexible language for transforming XML documents into various formats including HTML, XML, text, PDF etc. The input to XSLT transformations are XML Metadata Interchange (XMI) representations of models, which are based on XML syntax. XMI eases the problem of tool interoperability by providing a flexible and easily parsed information interchange format. In principle, a tool needs only to be able to save and load the data in XMI format. The conceptual design of SaveUML transformations is depicted in Figure 2.

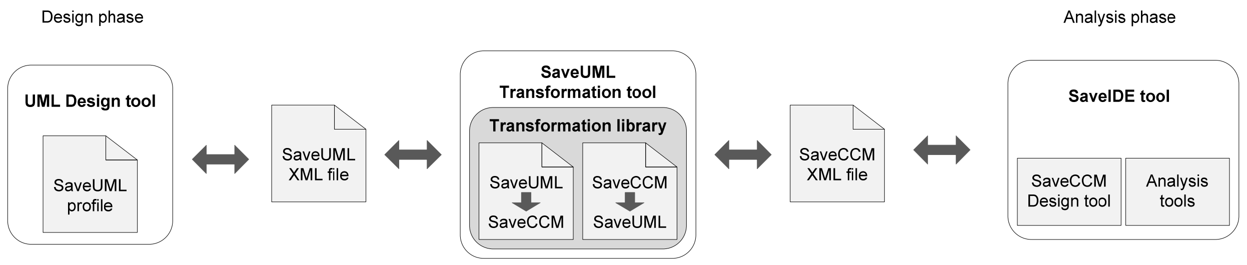

Figure 2. Conceptual design of the SaveUML transformations

The UML CASE tool is used for creating a UML model. Application of the SaveUML profile is necessary in order to create a UML model which can be transferred into a SaveCCM model. After designing the model, it is exported into an XMI file which is then used as the input to the transformation. The SaveCCM design tool is Save-IDE. For representing model information, SaveIDE uses several files which are compatible with XML and are used by the transformation tool to perform the SaveCCM to UML transformation. The tool uses the transformation library to perform translations. It contains XSLT style sheets for transforming from SaveUML into SaveCCM and vice versa. Input files based on XML are parsed through the XSL transformation style sheets and then XML-based output files, compatible with the desired tool, are generated.

Inspired by Visser's classification for program transformation [5] we classified SaveUML transformations as *language translation*, more precisely *migration*, as we transform between models specified in different languages at the same level of abstraction. After the transformation, the source model and the target model do not stay untouched but coexist and may evolve independently due to the development process. We implemented transformations in both directions, from UML to SaveCCM model and reverse. We implemented a prototype of the SaveUML transformation tool as a Java application.

5. AN EMPIRICAL COMPARISON OF SAVECCM AND SAVEUML TECHNOLOGIES

In previous sections we have described an approach for connecting two different modelling languages. A question arises on its usability in practical cases. Therefore we performed an experiment to verify the approach by comparing the modelling capabilities of both SaveIDE tool and RSM tool combined with SaveUML profile.

In this section we provide an overview of the experiment and its results, more detail can be found in [6].

5.1 Discussion and experiment objectives

Even though there are advantages of combining UML and SaveCCM in different development stages, how many benefits comes from these advantages and can they overwhelm the existing disadvantages and problems? In cases when usability of an UML profile is satisfying, and the expressiveness of UML extensibility mechanisms is sufficient for particular domain, then the need for building specialized tools is questionable. In these cases, designing an UML profile and using some of existing UML tools could replace a custom built tool.

There are several significant advantages that support this approach: (i) Using of already existing UML tools, which reduces time and cost spent on developing a domain-specific modelling environment; (ii) Any knowledge of and experience with standard UML is directly applicable; (iii) The UML profile is compatible with standard UML, thus any tool that supports UML can be used for manipulating models based on a UML profile. This brings the portability to models designed using SaveUML profile among many CASE tools.

However, the approach has some drawbacks: (i) A UML profile as an upgrade to basic UML can lead to an overly complicated model within an already complex UML specification and a modeller might get confused with extraneous UML semantics or modelling elements; (ii) Using standard UML notation, in which an existing shape corresponding a DSL (such as SaveCCM) element is used, could compromise the readability and clarity of the diagrams; (iii) A DSL is usually not used independently, but in combination with other tools, models and DSLs. In such a case the problem is not solved by expressing one DSL by UML extensions, but about a set of DSLs that should be mapped.

Our aim was to empirically evaluate our approach, with respect to development efficiency and ease of use. We wanted to compare the two modelling tools (SaveIDE and RSM along with SaveUML profile), to get feedback from the users and to ascertain advantages or disadvantages in using these tools.

With this experiment we tried to answer the following questions:

- Is using of SaveUML profile efficient with regard to time and efforts, in comparison to using SaveIDE?
- Do extraneous UML elements and semantics confuse developers and lead to an invalid or incomplete SaveCCM model?
- Which of technologies is more user-friendly and provide better user experience?

The given questions have more explorative character, so the results are shown mostly as descriptive statistics.

5.2 Conduction of the experiment

In this experiment 18 software engineering master students were given the task to design a model of a real-time system using either RSM (with SaveUML profile) or SaveIDE tool. As one of criteria for measuring development efficiency, we were monitoring efforts spent and quality of designed model in terms of its validity and detailness. After the modelling was finished we analysed models delivered by students, and students were given a questionnaire regarding their experience of working with the given modelling tool. Almost all of the students had some experience with UML. We conducted the study in a form of minutely described assignments for students with strictly defined deliverable deadlines. Our experiment was not conducted under controlled conditions in laboratory, but it is executed in the field under normal conditions i.e. in a real development situation. Except for the varying factor we wanted to study, which was a modelling technology, we controlled the qualification of the testers and an input for testing i.e. an example of a real-time system.

The actual study consisted of three phases. First we trained students in concerned technologies, then we conducted the experiment, and finally we let students to fill in a questionnaire and we analysed the results.

Training phase. The training phase lasted for three weeks. First two weeks were reserved for studying SaveCCM and UML (precisely UML component diagram). After two weeks students were given an exam which tested their knowledge. Based on the results of this exam we separated students into two groups, one that will use SaveIDE tool, and one that will use RSM tool with SaveUML profile. The disposition was made in a way to have two homogeneous groups with a comparable qualification range. Third week of training phase was intended for getting familiar with the tools by for modelling a simple system example.

Modelling phase. For the experiment we prepared a specification of Autonomous Truck Navigation (ATN) system demonstrated in [7]. This autonomous system is intended to navigate the truck to find and follow a straight black line drawn on the surface area. A model of the system, designed in RSM tool using SaveUML profile is shown in Figure 4.

Questionnaire. After the modelling was finished, participants completed a questionnaire regarding their user experience in the given tool. This questionnaire covered a number of subjects such as initial effort participants had to make to learn the technology, complexity of using the tool and clearness of graphical representation of modelling elements. Also there were questions about problems and bugs that encountered during their work, and several questions about different aspects of using SaveUML profile.

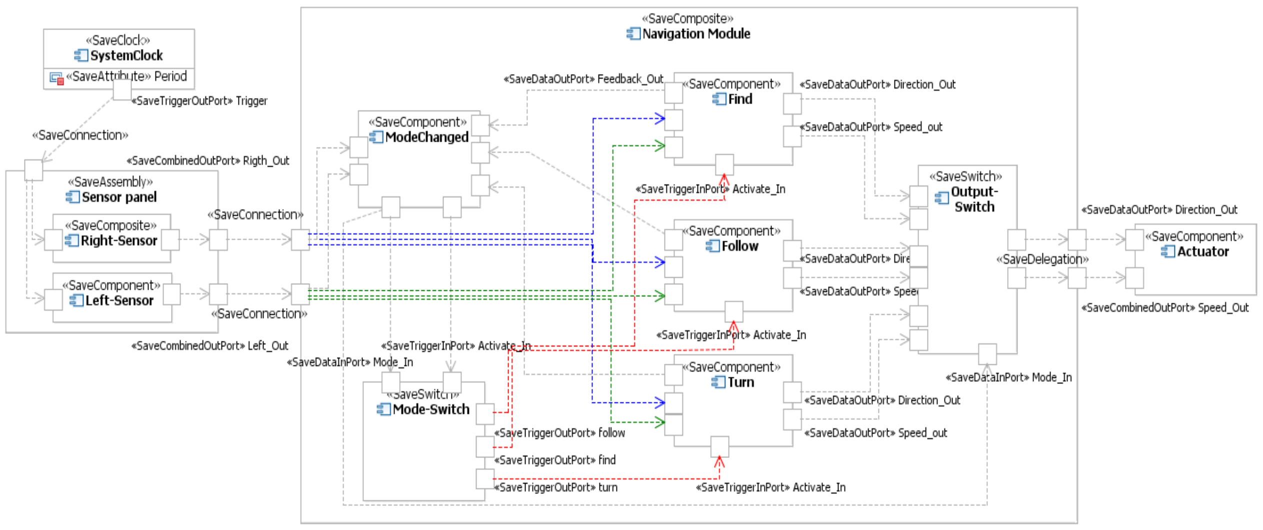
«SaveClock» SystemClock
«SaveAttribute» Period
«SaveTriggerOutPort» Trigger
«SaveConnection»
«SaveCombinedOutPort» Rigth_Out
«SaveAssembly» Sensor panel
«SaveComposite» Right-Sensor
«SaveComposite» Left-Sensor
«SaveConnection»
«SaveConnection»
«SaveCombinedOutPort» Left_Out
«SaveComposite» Navigation Module
«SaveDataOutPort» Feedback_Out
«SaveComponent» Find
«SaveDataOutPort» Direction_Out
«SaveDataOutPort» Speed_out
«SaveComponent» ModeChanged
«SaveTriggerInPort» Activate_In
«SaveComponent» Follow
«SaveDataOutPort» Dire
«SaveDataOutPort» Spee
«SaveTriggerInPort» Activate_In
«SaveSwitch» Output-Switch
«SaveDelegation»
«SaveDataOutPort» Direction_Out
«SaveComponent» Actuator
«SaveCombinedOutPort» Speed_Out
«SaveDataInPort» Mode_In
«SaveTriggerInPort» Activate_In
«SaveDataInPort» Mode_In
«SaveSwitch» Mode-Switch
«SaveComponent» Turn
«SaveDataOutPort» Direction_Out
«SaveDataOutPort» Speed_out
«SaveTriggerOutPort» follow
«SaveTriggerOutPort» find
«SaveTriggerOutPort» turn
«SaveTriggerInPort» Activate_In

Figure 4. A simplified model of Autonomous truck navigation system

5.3 Results

By analysing the results of the experiment, our goal is answer the three questions raised above.

Which of technologies is more user-friendly and provide better user experience? The initial effort participants had to make to get to know with the tool was slightly different for the two tools. As it can be seen in Figure 3a, group using SaveUML had more problems when starting to use the tool. One of the possible reasons was that RSM is a more complex tool and offers a lot of possibilities which confused them. However, the SaveUML group also reported more improvement after having some training with the tool (Figure 5b).

Considering the complexity of using the tool, users of SaveUML found project management, defining properties of model elements and adding new elements to the model a bit more intuitive than the SaveIDE users (figure 5c). This is not very surprising for the first two as the RSM tool is more advanced and these actions are common to all UML modelling tools.

Workspace organisation and overall complexity of using the tool were graded very similar by both groups. SaveUML users also reported better assistance (automated procedures, offering default values etc.) from the tool, which is not surprising considering that RSM is a professional tool.

As SaveIDE uses all custom graphics and SaveUML only the default UML graphics, it was expected that the SaveIDE group would report much better readability of models than the SaveUML group. Although the SaveIDE group reported a better readability, the difference between the SaveIDE and SaveUML groups is not as large as expected, as it can be seen on 7c. The comments given by participants indicate that some flaws in the graphical representation in the custom tool have a big impact on experience of graphical environment.

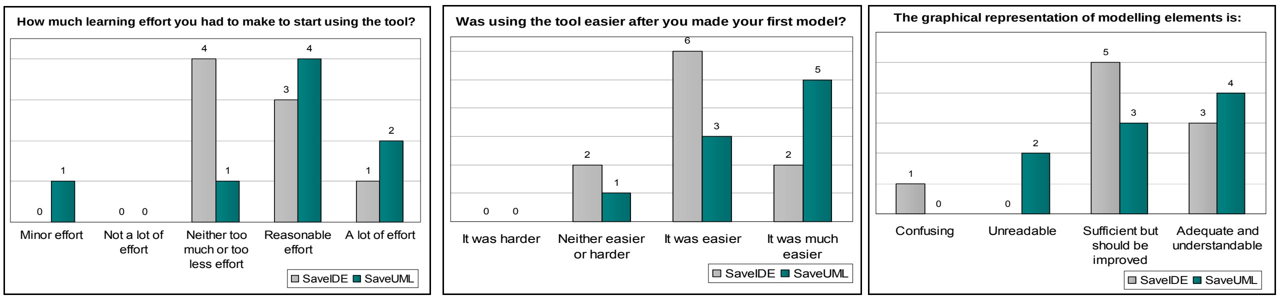

Figure 3. a) Initial effort for using the tool; b) Effort after adaption period: c) Perspicuity of graphical representation

Users of SaveUML spent in average 8.22 hours and Users of SaveIDE 6.28, so somewhat less hours.

Do extraneous UML elements and semantics confuse developers and lead to an invalid or incomplete SaveCCM model? The group using SaveUML found it harder to get accustomed with using the RSM tool to model SaveCCM. From their comments we concluded that all the features that the RSM tool provides initially confused them. In addition, UML profile extends UML, but does not repress the usage of non-extended part of UML. The availability of various UML elements which do not belong to SaveCCM domain was the cause of most mistakes that students from SaveUML group made in their models. The common mistakes were (i) Using of wrong relationships for connecting model elements; and (ii) Not setting the properties (e.g. data type of ports) through the tagged value mechanism (setting of properties using tagged values is not as intuitive as setting of standard properties).

However, after the training phase almost all of the participants got acquainted with the tool, and these mistakes were rare.

Is using of SaveUML profile efficient with regard to time consumption, in comparison to using SaveIDE? The usage of SaveUML profile had somewhat negative impact on time consumption. This inefficiency arises from the fact that it takes many steps to accomplish a simple operation. For example to add a SaveCCM component to the model, first a UML component has to be added to the model, then an appropriate stereotype from the UML profile has to be applied and finally component attributes can be set.

Overall results. By this experiment we have indicated usefulness of the approach of adoption of a general purpose tool instead of a DSL (or in this concrete example of creating UML profile SaveUML as an alternative to SaveIDE. This experiment was focused on feasibility and usability of the design phase of component-based systems. We have not tested usability in using the specifications documents and means for exchanging information between users. Neither have we evaluated a larger scope of the lifecycle that includes analysis and verification part which requires repetitive transformation between the tools. The approach cannot be generalised in the sense that such approach is always better or feasible, but the experience indicates this possibility with pointing out the possible challenges.

6. RELATED WORK

Many researchers have tried to accomplish linking of UML with some DSLs. For instance, Polak and Mencl developed a mapping from UML 2.0 to SOFA and Fractal research component models [8]. The approach also uses UML profiles for designing UML models and a tool prototype generates SOFA and Fractal source code from UML model. Contrary to the SaveUML profile, the UML profile they create is used only to define new UML metaclasses using stereotypes and tagged values, while constraints (defined by OCL) do not exist.

The work by Malavolta et al. [9], is not limited to particular modelling languages. The automated framework called DUALLy creates interoperability among various ADLs, as well as UML. DUALLy is partitioned to two abstraction levels, separating meta-model definition process and system development. The transformations between languages are not done directly but there is a central *A_0 model* using as a intermediate step of every transformation. A_0 is a UML profile and it represents a semantic core set of architectural elements (e.g. components, connectors, behaviour). It provides the infrastructure upon which to construct semantic relations among different ADL and acts as a bridge among architectural languages. The disadvantage of this approach is that defined mappings are not injective, thus the unique reverse transformation is not ensured.

7. CONCLUSION

In this paper we presented a simple approach for achieving modelling language interoperability between UML and a domain specific language SaveCCM. The main idea is creating a UML profile to allow developing UML models with domain-specific semantics. Further, a transformation tool for such model to SaveCCM is implemented, which makes it possible to use analysis of timing and other properties. The transformation is achieved using XML representations of models as an input for XSLT style sheets. The proposed approach fosters combining of GPL and DSL at different design stages. Some of the benefits are making a good use of advantages of both languages which improves design productivity, portability of the model as well as already mentioned standardization. We have validate the feasibility of the approach and usability of the UML profile-based tool in comparison to SaveCCM tool, and found that in spite of a quite different characteristics of models the approach, adopting a general purpose tool in this case was feasible since in the experiment similar results were achieved by both tools.

8. REFERENCES

[1] Akerholm, M., Carlson, J., Fredriksson, J., Hansson, H., Håkansson, J., Möller, A., Pettersson, P., and Tivoli, M. 2007. The SAVE approach to component-based development of vehicular systems. *J. Syst. Softw.* 80, 5 (May. 2007), 655-667. DOI= http://dx.doi.org/10.1016/j.jss.2006.08.016

[2] Åkerholm, M., Carlson, J., Håkansson, J., Hansson, H., Nolin, M., Nolte, T., and Pettersson, P. 2007. The SaveCCM Language Reference Manual. MRTC report ISSN 1404-3041 ISRN MDH-MRTC-207/2007-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, January, 2007.

[3] OMG 2007. Unified Modelling Language Superstructure Specification. Version 2.1.1, February, 2007., http://www.omg.org/uml/

[4] W3C 1999. XSL Transformations (XSLT). Version 1.0, W3C Recommendation, November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[5] Visser, E. 2001. A Survey of Strategies in Program Transformation Systems. Electronic Notes in Theoretical Computer Science, eds. Gramlich and Lucas, vol. 57, Elsevier, 2001.

[6] Petričić, A., Lednicki, L., Crnkovic, I., 2009. An empirical comparison of SaveCCM and SaveUML technologies, http://www.mrtc.mdh.se/publications/1621.pdf, MRTC, Mälardalen University. March, 2009.

[7] Sentilles, S., Pettersson, A., Nyström, D., Nolte, T., Pettersson, P., and Crnkovic, I., 2009. Save-IDE - A Tool for Design, Analysis and Implementation of Component-Based Embedded Systems, Proceedings of the Research Demo Track of the 31st International Conference on Software Engineering (ICSE), Vancouver, May, 2009

[8] Mencl, V., and Polak, M. 2006. UML 2.0 Components and Fractal: An Analysis. 5th Fractal Workshop (part of ECOOP'06), July 3rd, 2006, Nantes, France, Jul. 2006.

[9] Malavolta, I., Muccini, H. and Pelliccione, P. 2008. DUALLY: a framework for Architectural Languages and Tools Interoperability. 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE2008). September 15-19 2008 L'Aquila, Italy. IEEE Press.