Models Transformation between UML and a Domain Specific Language

Ana Petričić, Ivica Crnković
Mälardalen University
Mälardalen Research and Technology Centre
PO Box 883, SE-721 23 Västerås, Sweden
+46 21 {15 17 28, 10 31 83}
{ana.petricic, ivica.crnkovic}@mdh.se

Mario Žagar
University of Zagreb
Faculty of Electrical Engineering and Computing
HR-10000 Zagreb, Unska 3, Croatia
mario.zagar@fer.hr

ABSTRACT
As complexity of software systems is increasing, using a proper modelling language for designing and analysing a system is becoming more and more important. Over the recent years there is a tendency for using domain-specific languages which enable expressing design solutions in the idiom and the level of abstraction of the specific problem domain. Since a design process passes through different levels of abstractions and different properties of systems are being modelled, different modelling languages are used. While this approach enables an efficient and accurate design, it suffers from a problem of transformation between the models. This paper addresses a challenge of transformation between UML, a modelling language widely used, to a domain-specific language SaveComp component model (SaveCCM) intended for real-time embedded systems. In this paper we discuss a possible solution for achieving interoperability between SaveCCM and UML. The challenge of a transformation is to keep all necessary information including the semantics of the models. The paper presents the strategy for the transformation, its implementation and an analysis of the results.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *modules and interfaces,*

General Terms
Design, Languages

Keywords
Software component models, models transformation, UML, domain-specific languages

1. INTRODUCTION
Over the recent years, as information technologies have evolved, the role of models is becoming essential for dealing with the numerous aspects involved in system's development and maintenance processes. Therefore, it is particularly important to use an appropriate modelling language when designing the system in order to improve efficiency at the design phase, to provide analyzability as well as to decrease the time-to-market.

A number of various modelling languages exist nowadays, each of them using different abstractions and notations. However, there is a continual aspiration for defining a universal modelling language in order to standardize notations and accomplish tool interoperability. With these objectives in focus, the Object Management Group (OMG) developed the Unified Modeling Language (UML) [11] which became a *de facto* standard for modelling. UML is well suited for modelling the most common aspects of software architecture, but it fails when it comes to more specific cases e.g. modelling quality attributes of a system and its components or a time analysis.

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 and then to apply that profile in addition to general UML. UML profile specializes the standard UML concepts by defining constraints on those concepts appending them a unique domain-specific interpretation. The creation of UML profiles is a way of producing what are now referred as Domain Specific Languages (DSL).

Even though UML presents a good starting point for creation of DSL, this does not imply that any DSL should be realized using UML profiles. There are many cases where UML may lack the requisite foundation elements that can be cast into corresponding DSL elements. For example, UML does not provide solutions for handling of typical properties important in embedded systems and safety-critical systems, such as resource efficiency, predictability and safety. For such cases there is a need for specialized modelling languages which will provide more expressiveness at the design time and efficiency in analysis and testing.

One of such DSL is the SaveComp Component Model (SaveCCM) [2], a research component model intended for embedded control applications in vehicular systems. SaveCCM is a simple model in which flexibility is limited to facilitate analysis of real-time characteristics and dependability. SaveCCM as a domain specific language, lacks the power of a general purpose language (GPL) such as UML, but is very productive for designing safety-critical subsystems responsible for controlling the vehicle dynamics, including power-train, steering, braking, etc. A disadvantage of a DSL is paradoxically, its specificity – it can be inappropriate for modelling different properties and it may require additional efforts to be used. In particular it can provide

SERPS'08, November 4-5, 2008, Karlskrona, Sweden.

obstacles in communication of the design decisions between different stakeholders. Therefore we found a motivation for joining of GPL and DSL, in order to provide the possibility of creating a model of a system in one language and then transforming it into another language, thus making a good use of advantages of both languages. This approach can combine two different modelling languages in different design stages. In an early design stage, a model of the system can be designed using UML which provides the user a great efficiency of the development process. After creating a model in UML the subsequent step is transforming the UML model of the system into SaveCCM model for further analysis of non-functional properties that can not be tested in UML. The latter design phase takes advantages of SaveCCM which is intended for modelling of safety-critical embedded systems and testing of run-time properties, namely, timing and resource usage characteristics.

In order to create a link between UML and SaveCCM and to enable the development of UML models which could be transformed to a SaveCCM model, a mapping from UML to SaveCCM must be specified. This mapping can be achieved using one of the UML extensibility mechanisms mentioned above – creating a UML profile. Due to the universality of UML and a strict syntax of SaveCCM, restrictions on UML semantic and syntax have to be imposed, hence the application of the profile is necessary for developing of a model which can be transformed to SaveCCM domain.

The goal of our work is to obtain a simple solution for achieving interoperability between two modelling languages. Further the goal is to analyze the feasibility of the approach in terms of full and unique transformation of models in both directions.

In this paper we describe a formal way of representing the SaveComp component model using the UML 2.0 component model. This is achieved through a specification of SaveCCM elements in UML in the form of a UML profile named SaveUML. Furthermore, a possible solution for an implementation of a model transformation is proposed. The discussion on the usability of created UML profile and the transformations is provided.

The rest of the paper is organized as follows: After a brief description of SaveCCM in section 2, in section 3 and 4 we present the SaveUML profile and a design of transformation of system models between UML and SaveCCM. Section 5 discusses the characteristics and applicability of the profile and the transformations. Finally, section 6 presents related work and section 7 gives our concluding marks.

2. THE SAVECCM OVERVIEW

SaveCCM is anticipated for designing safety-critical subsystems responsible for controlling the vehicle dynamics. It uses the component-based development (CBD) approach, thus enabling assembling of systems out of components which are already developed and prepared for integration. Contrary to many of the current component technologies, SaveCCM focuses on predictability and analysability more than on flexibility. In addition SaveCCM supports the development of resource-efficient systems. SaveCCM technology provides a support for a design of subsystems and analysis of timing properties built in an integrated development environment SaveIDE.

In SaveCCM, systems are built from interconnected elements with well-defined interfaces consisting of input and output ports. 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 (execution can be initiated by a clock or external elements). Due to this separation, the resulting design is analysable with respect to temporary behaviour, thus allowing analysis of schedulability, response time, execution time etc.

In addition to ports, the interface of an element may contain quality attributes each associated with a value and possibly a confidence measure. These attributes hold the information about the worst case execution time, reliability estimates, safety models, etc. The quality attributes are used for analysis, model extraction and for synthesis.

The main architectural elements in SaveCCM are:

- *Components*, which are the basic units of encapsulated behaviour with a functionality that is usually implemented by a single entry function in C. Besides an entry function, each component is defined by associated ports and optionally quality attributes.
- *Switches*, which provide facilities to dynamically change the component interconnection structure (at configuration or run-time); this allows a conditional transfer of data or triggering between components.
- *Assemblies*, which provide means to form aggregate components from sets of interconnected components and switches.

Contrary to components, assemblies and switches can not be activated, they respond instantly at the arrival of data or trigger signal on some of the input ports and they can both be considered as special types of components, however due to the difference in semantics they are treated as separate elements.

SaveCCM also provides a 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 an entry function.

A subset of the UML 2.0 component diagram is adopted as graphical representation language. The interpretation of the symbols for provided and required interfaces, and ports are somewhat modified to fit the needs of SaveComp. The symbols used are depicted on Figure 1.

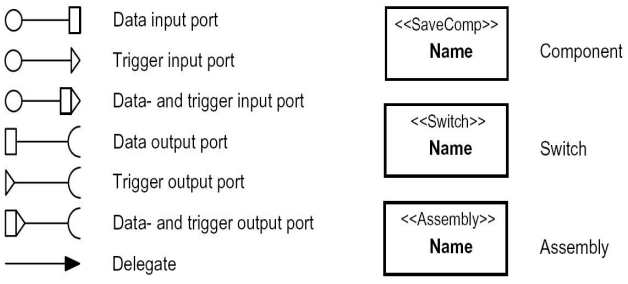

Figure 1. Graphical notation of the SaveComp component model

The SaveCCM *run-time framework* provides a set of services, such as communication between components, component execution and control of sensors and actuators.

SaveCCM as a domain specific language brings some valuable advantages:

- Enables expressing design solutions in the idiom and the level of abstraction of the embedded systems problem domain.
- Enhances quality, reliability and maintainability of the model.
- Provides the possibility of model reusability, which is one of the main concepts and vantages of CBD approach.
- Improves productivity of the development process.
- Allows validation and testing at the domain level.
- Reduces the time needed for developers to learn the modelling language, since it uses similar concepts and terms.

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 [2] and [1][3][5].

3. THE SAVEUML PROFILE – A UML SPECIFICATION OF SAVECCM

In this section we provide a comparison of UML and SaveCCM and propose a set of UML extensions to support the dependability analysis of real-time and embedded systems. Since our goal is to be compliant with the SaveComp component model, our main guideline is the SaveCCM language specification [2]. Furthermore, the profile notation and design decisions were specified to be as much as similar as the ones defined in the SaveIDE design tool [9][14], with a view to avoid inconsistency.

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 and we call this subset a UML 2.0 component model.

3.1 Differences between SaveCCM and UML 2.0 component model

When contemplating the differences between UML and SaveCCM, the cardinal difference that needs to be emphasized is a distinction between domain specific and general purpose modelling language. A DSL is created specifically to solve problems in a particular domain and can be used to directly model concepts in that domain using an appropriate level of abstraction. One of the causes that make DSL specific and very powerful is the ability to express relationships between concepts of the domain. In basic UML, the relationships are applied generically (e.g. a UML association can relate any type of class). In SaveCCM many relationships are constrained (e.g. two components in a component-based system can only be connected if the data types they exchange are compliant).

UML provides a great efficiency of the development process. On the other hand, experience has shown that for many embedded system domains efficiency in run-time resources consumption and prediction of system behaviour at least as important as efficiency in the software development. Contrary to UML, SaveCCM focuses on predictability and analysability more than on flexibility.

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 can not be found in UML.

- The distinction between data transfer and control flow, which is achieved by distinguishing two kinds of ports; data ports and trigger ports.
- When considering interfaces, UML has created a special metaclass for this purpose. On the other part, in SaveCCM the functional interface of every modelling element is defined by a set of ports associated to the element and optionally, quality attributes.
- One of the important capabilities of SaveCCM is model analysis and verification. SaveCCM uses quality attributes for defining non-functional properties of a component and a system. Each quality attribute is associated with a value and possibly a confidence measure.
- In order to provide run-time model analysability, SaveCCM defines the execution model of active model elements. The execution model is rather restrictive; the basis is a control-flow (pipes and filter) paradigm in which executions are triggered by clocks or external events, and where components have finite, possibly variable, execution time. The component execution semantics is defined by a sequence of activities: start by trigger, read, execute, and write.

3.2 SaveUML profile overview

We have chosen to develop a UML profile which is a standard UML extensibility mechanism. UML profiles provide an ability to customize the language by adding new building blocks, creating new properties and specifying new semantics in order to tailor UML to a specific problem domain in a controlled way. Considering that UML profiles are a standard UML extension mechanism and are therefore a part of UML's metamodel, they are as widely recognized as UML itself and should be supported by all standard modelling CASE 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 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 the unique transformations between the UML and SaveCCM models preserving the SaveCCM semantics.

The process of defining SaveCCM language elements using UML 2.0 elements 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 component models and identify compatible elements. In addition we defined the transformation rules for all elements from SaveCCM 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 modelling elements. This phase included defining a set of constraints that covered the SaveCCM semantics as well as additional restrictions that had to be imposed upon the UML model to prohibit the usage of elements and concepts that do not fit in the SaveCCM semantics.
3. Translation of previously identified elements during which a suitable UML element is found for every SaveCCM language elements and it was then further customized through the use of necessary stereotypes, properties and constraints.

The process result is the profile specification which describes the generated UML profile as a list of its stereotypes, the basic UML elements they extend and the source SaveCCM elements they represent. The diagram of the SaveUML profile is depicted in Figure 2.

Essentially, the concepts of the metamodel are reflected onto stereotype attributes and constraints. 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, after a comprehensive analysis of both component models, 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. These design decisions are presented below.

Components

Since SaveCCM is intended for modelling of component-based systems, the basis for main architectural elements in SaveCCM is a component. SaveCCM introduces three main architectural elements; component, assembly and switch. In addition, three subtypes of SaveCCM component are defined; clock, delay and composite component. All together, this makes six different kinds of components in context of CBD. UML 2.0 component model provides only one kind of a component. For the needs of SaveCCM it is necessary to distinguish between six types of components, therefore it is required to define six virtual metaclasses – stereotypes that will extend the UML `Component` metaclass. This will allow applying those stereotypes to components within the user model to distinguish between

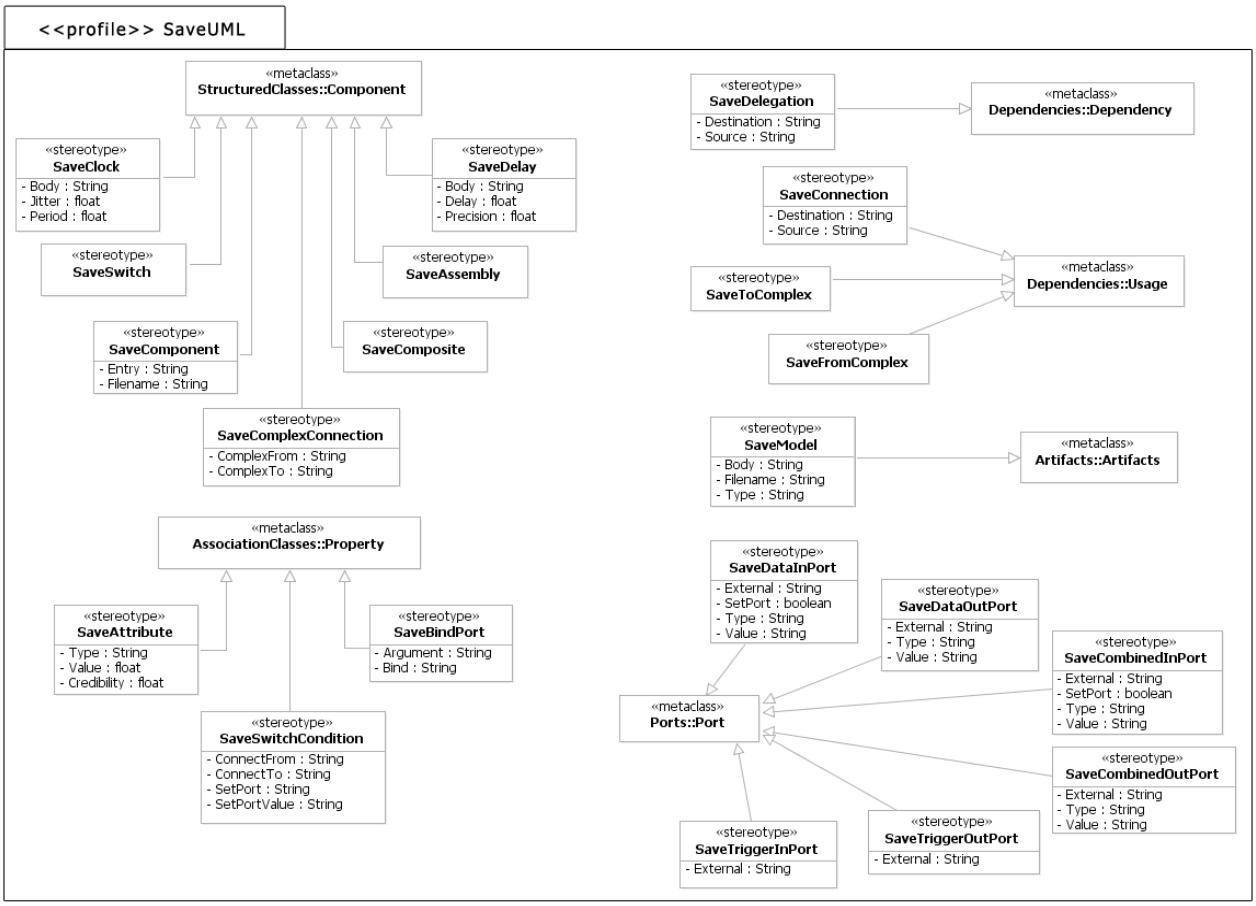

Figure 2. Diagram of the SaveUML profile

SaveCCM architectural elements.

*Subcomponent*s

SaveCCM offers two elements that may have an internal structure defined, assembly and composite component. In UML 2.0, there are two ways of specifying an internal structure of a component: using metaclass `Property` either using metaclass `PackageableElement`.

The advantage of the first approach is defining of subcomponents outside the owning element at one place, which can then be referred to as a type of the part (in UML internal sub elements that are defined using metaclass `Property` are called *parts*). This way, as many parts as needed can refer to the same component at the same time. The changes made to the component will reflect to all parts referring that component. However, this approach has a drawback. Since a metaclass `Property` is not a subtype of an `EncapsulatedClassifier`, it may not have an internal structure. This means that it is not possible to nest the components to arbitrary depth. A component can have only one-level internal structure (the only solution to this problem is to define internals of a subcomponent outside the owning component). In addition, using `Property` for defining subcomponents is not semantically correct regarding the CBD.

Considering disadvantages of using the `Property` metaclass, the latter approach – using `PackageableElement`, was chosen. This method enables defining a hierarchical composition of components and its nested subcomponents to an arbitrary depth at one place with no need to define internals of subcomponents outside the owning component. Such a definition of subcomponents is called *embedded definition of components*.

Interfaces

In SaveCCM the functional interface of every modelling element is defined by a set of ports associated with the element. UML 2.0 provides an `Interface` metaclass for this purpose and supports two ways of specifying provided and required interfaces. Interface provides a way to partition and characterize groups of properties and operations that a component possesses. Because of semantic differences of interface in the SaveCCM and UML, we decided not to use UML interfaces in SaveUML profile. It is supposed that when modelling a user model in UML using the SaveUML profile, the interface of a component will be determined by its ports, as it is done in SaveCCM.

3.3 Using OCL for user model validation

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) [10] which is an OMG standard language used to describe expressions on UML models. Besides the SaveCCM semantics, OCL constraints are also defined to enforce the restrictions on UML model to proscribe the usage of UML concepts that do not have equivalent elements within SaveCCM. Constraints are intended to be used for validating a user model in order to ensure that the model is valid in consideration of 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 the following table. For each group number of constraints within the group is displayed.

Table 1. Constraints implemented in SaveUML profile

Constraint group	Description	Count
Restrictions on UML		56
forbidden connections	Restrictions on UML 2.0 considering using various types of connectors (only `Usage` and `Dependency` are used within the profile). Also, connectors should not connect elements directly etc.	17
using interfaces	As discussed, using interfaces is not allowed within the SaveUML profile, these constraints are dealing with this issue.	12
substructure definition	As mentioned above, internal structure of an element may only be defined using packaged elements. Further, the only allowed packaged element is a `Component` (it is the base for all SaveCCM architectural elements). Also, some SaveCCM elements are not allowed to have internal sub-structure at all.	6
number of stereotypes	Even though UML has this option, in SaveUML profile, one element can have only one stereotype applied.	21
SaveCCM semantics		61
owning attributes	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 (clock and delay).	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	Since SaveCCM offers two	23

SaveCCM elements	kinds of connections, each connector must have an appropriate stereotype applied. Also, depending on the connection type, cyclic connections are forbidden or allowed. Finally constraints ensure conformance of the connected ports (their types and directions).	

Generally speaking, we found that identification and definition of OCL constraints is the major part in creating a UML profile. An important feature in OCL is using navigation expressions, therefore it is expected that the developer is familiar with the model, in our case with UML itself. We found that this can be a challenging task for non-experienced UML user, as UML is a complex language with a large number of elements and various diagram types. Due to this reason, it requires experience in using OCL and it takes a considerable amount of efforts to identify and specify all constraints.

Also an important part in using OCL is the tool support. The profile specification provided in previous section is a general profile description and we have chosen IBM Rational Software Modeler (RSM) [6] for implementing the profile prototype. RSM is a commercial product of IBM supporting standard UML 2.0 functionality. It is built on top of the open and extensible Eclipse platform that leverages several open industry standards to provide a significant level of extendibility. RSM provides a standard interface for the definition of profiles consequently stored in XML. OCL is also supported for constraint authoring. The tool facilitates constraint checking using the "Run Validation" option. Furthermore, we used another rewarding feature that RSM offers; two different kinds of constraints. In prototype we implemented, constraints defined within the profile are divided into two groups, constraints with *live validation* and constraints with *batch validation*. Constraints with *batch validation* are checked when the user runs a validation action. An example is a constraint which requires that all ports owned by a component have an appropriate stereotype applied. To meet this demand, it is necessary to take a two-step process, therefore this constraint can not have a *live validation* (the constraint would be violated after the first step). Constraints with *live validation* are checked every time the model element, to which the stereotype is applied, is modified. E.g. a constraint that suppresses using any connectors directly on a component (in SaveCCM no connections except the ones explicitly captured by the ports are allowed). If a constraint with *live validation* is violated an immediate notification arises. This distinction of constraint categories and their handling, facilitates model validation and allows the validation to be achieved already during design time.

4. SAVEUML TRANSFORMATIONS

The purpose of creating UML profile is twofold; i) provide ability of using UML to model SaveCCM systems in an early phase of the design, and ii) enable bidirectional transformation from UML to SaveCCM.

In this section we describe the conceptual idea of the transformations between UML and SaveCCM models, named SaveUML transformations. Further we present a case study example used for verifying the transformations.

4.1 Conceptual design

The SaveUML transformations categorisation was inspired by Visser's classification for program transformation [15]; according to this classification there are *language translation* and *language rephrasing*. In the former, a model is transformed into a model of a different language, i.e., a different model, and in the latter, a model is changed in some way, which may involve producing a new target model with the changes or changing the existing source model. The SaveUML transformation fits into the first type, as it transforms between two different models, UML and SaveCCM. Furthermore, like in [15], *language translation* can be sub-divided into *migration*: a model is transformed to another one at the same level of abstraction; *synthesis*: a model is transformed to another one at a lower level of abstraction; and *reverse engineering*: a model is transformed to another language at a higher level of abstraction. As the UML model which is transformed is created using the appropriate UML profile, the level of abstraction is not changed by the transformation which places the SaveUML transformation within the migration category.

The transformation approach is based on using the eXtensible Stylesheet Language for Transformations (XSLT) [16]. 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. However, this is not yet accomplished, and there are minor differences in generated XMI model representations between various tools. Therefore the transformations are still tool-dependent.

The conceptual design of SaveUML transformations is depicted graphically in Figure 3.

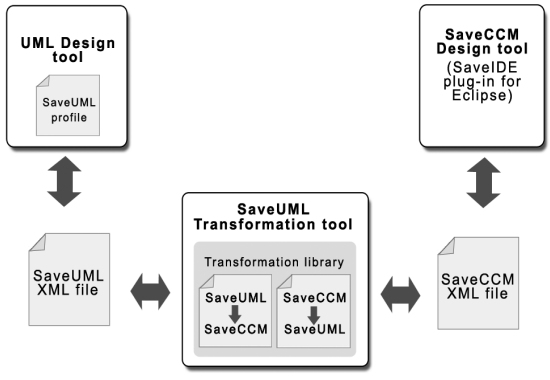

Figure 3. Conceptual design of the SaveUML transformations

The UML CASE tool is used for creating a UML user model. By applying the SaveUML profile, UML elements are stereotyped for

modelling SaveCCM elements. As already mentioned, the application of the profile is necessary in order to create a 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 for the transformation. The SaveCCM design tool is Save-IDE. SaveIDE uses several files for representing model information. Those files 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.

The transformation tool we developed as a prototype can be used either as an Eclipse plug-in or as a standalone application and performs transformation in both directions, UML to SaveCCM and SaveCCM to UML models.

4.2 Transformation example: an adaptive cruise controller

The Adaptive Cruise Controller (ACC) [3] has been a recurring example throughout the development of SaveCCM. The purpose of this running case-study has been to continuously evaluate and improve the component model. We demonstrate a simple design of ACC [5] as an example for verifying the transformations.

The ACC system helps the driver to keep a desired speed and a safe distance to a preceding vehicle. The ACC automatically adapts the distance depending on the speed of the vehicle in front, while keeping the gap large enough to avoid collisions. The SaveCCM model of ACC system is presented on Figure 4 and Figure 5, and it can be divided to three parts: input, control and actuate.

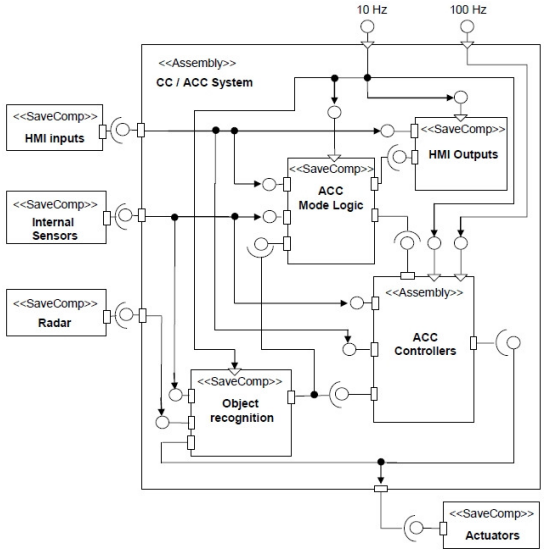

Figure 4. ACC system design

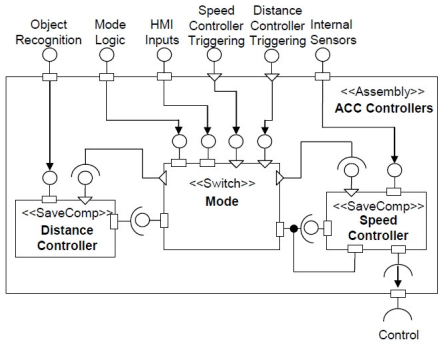

Figure 5. ACC controllers design

The ACC system is designed as a SaveCCM assembly ("ACC System" on Figure 4) built from three basic components ("Object recognition", "ACC Mode Logic" and "HMI outputs") and one sub-assembly ("ACC controllers"). Internal design of "ACC controllers" sub-assembly is provided by two components ("Distance Controller" and "Speed Controller") and one switch ("Mode"). The detail description of functionality of those elements can be found in [1][3][5].

To demonstrate the SaveUML transformation tool and to illustrate the usage of SaveUML profile as well as to verify the transformations, we created an ACC system UML model using SaveUML profile. The model is presented on Figure 5.

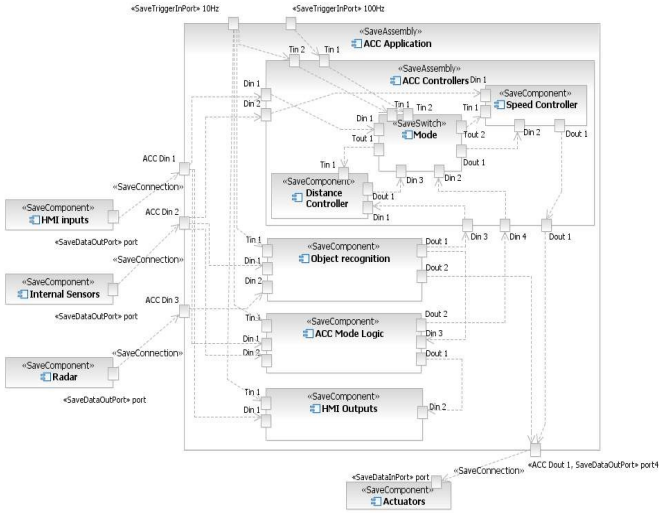

Figure 5. UML model of ACC system

SaveUML transformation tool transforms this model into a SaveCCM model consisting of one *saveccm* file. The screenshot of this file form SaveIDE environment is shown on Figure 6. After the transformation, using SaveIDE tool, it is possible to generate diagram files for visual presentation of the model.

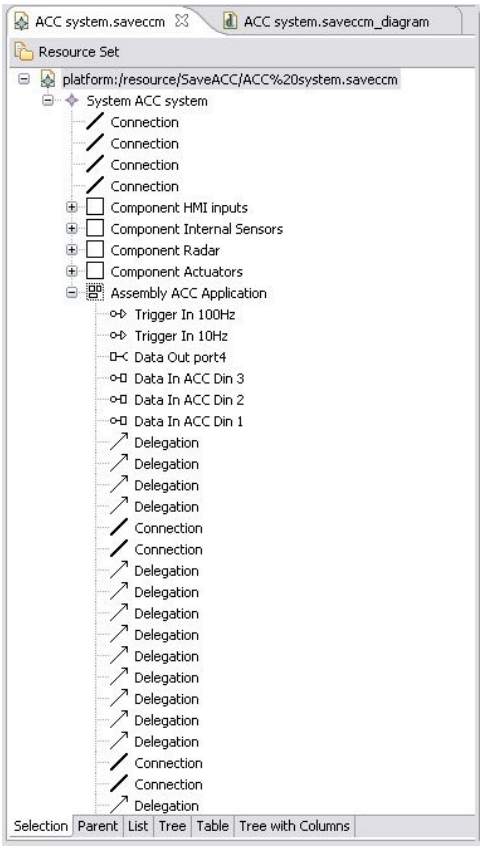

Figure 6. ACC system model after transformation

5. DISCUSSION

In this paper we have described one possible approach for connecting two different modelling languages. However a question arises on usability in practical cases of both the created profile and the transformations. Even though there are numerous advantages of combining UML and SaveCCM languages in different development stages as well as for different purposes, how many benefits comes from those advantages and can they overwhelm the existing disadvantages and problems? In this section we reveal and discuss those advantages and disadvantages of using the SaveUML profile and transformations.

The UML profile for SaveCCM can be used to develop models describing systems from SaveCCM application domain. It may appear that defining the SaveUML profile is just reducing the UML scope and specifying a component model which is an UML subset with modified terminology. Yet the SaveUML profile brings additional semantics through its constraints and introduces SaveCCM abstraction level.

The SaveUML profile brings SaveCCM semantics to UML, so any knowledge of and experience with standard UML is directly applicable. The DSLs built using profiles are built from the ground up, so the modeller is not confused with extraneous UML semantics or modelling elements. This is useful for users accustomed to UML or if UML is a standard modelling language within a company, so there is no need to switch to SaveCCM.

The UML profile is compatible with standard UML, thus any tool that supports UML can be used for manipulating models based on the UML profile. This brings the feature of portability to the SaveUML profile among many CASE tools. Contrary to UML models with SaveCCM semantics, SaveCCM models created with the SaveIDE tool can only be managed by SaveIDE.

A number of UML CASE tools exist that provide a user friendly interface and functionality which is a behalf on using UML and the SaveUML profile instead of genuine SaveCCM in the SaveIDE modelling environment. Although the tool and UML itself provide a great efficiency at design time, using profiles is not as efficient as using basic UML which is one of the disadvantages of SaveUML profile. This inefficiency rises 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. Consequently it is ambiguous if using the profile brings any efficiency or not. Our opinion is that this depends on the modelling tool used and its support for UML profiles.

Having in mind that UML profiles are an upgrade to basic UML, this can lead to an overly complicated model within what has been described by many in the industry as an already complex specification. Using standard UML notation, in which an existing shape corresponding to SaveCCM element is reused, could compromise the readability and clarity of the diagrams.

Finally, it is not an easy task to find a proper CASE tool that has a good and accurate support of UML profiles, often this support is poor or is missing entirely.

Combining UML and SaveCCM languages and using transformations between them implies existence of at least two models which represent the system design. 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. Therefore, in order to preserve a coherent description of the system in both models, it is necessary to propagate certain changes between them. We implemented transformations in both directions, from UML to SaveCCM model and reverse, having in mind this request. Reverse transformation, i.e. transforming the model from one language to another and back to the starting language, should produce a model equivalent to the initial one. SaveUML profile already provides a one-to-one mapping from UML to SaveCCM. In addition, models are transformed at the same level of abstraction which makes these transformations injective. The transformation process itself comes to transforming from one XML representation of a model to another XML file. Therefore, the request for a unique transformation is fulfilled.

6. RELATED WORK

Many researchers have tried to accomplish linking of UML with some DSL. For instance, Polak and Mencl developed a mapping from UML 2.0 to SOFA and Fractal research component models [8][12]. 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 created is used only to define new UML metaclasses using stereotypes and tagged values. Constraints on

model semantics are implemented in the source code generator. OCL is a specification language and has some restrictions, and in many tools the OCL support is weak. From this point of view, implementing constraints in the source code generator is reasonable. On the other hand, if constraints are not present in the profile, the user can not validate the model during design-time, which reduces design efficiency. Also, OCL is a standardized language promoting interoperability between modelling tools. Standard profile definition process in most of the tools supports storage of the profile in an XML file using XMI including OCL constraints, which can consequently be imported in another UML 2.0 modelling tool. This can not be applied for the suggested approach as well.

The work by Malavolta et at. [7], is not limited to particular modelling languages. The automated framework called DUALLy creates interoperability among various Architecture Definition Languages (ADL), as well as UML. DUALLy is partitioned to two abstraction levels, separating meta-model definition process and system development. At the meta-modelling level model driven engineers provide a specification of the architectural language in terms of its meta-model or UML profile Also, they define semantic links between the meta-models and afterwards DUALLY automatically instantiates these semantic links into model-to-model transformations. At the modelling level, software architects use generated transformations for translating the system models among preferred languages. A significant feature of DUALLy is its ″star architecture″. The transformations between languages are not done directly but there is a central *A0 model* using as a intermediate step of every transformation. A0 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 ″star architecture″ decreases the number of semantic links that needs to be defined among modelling languages, as it is necessary only to define relationships between the concerned language and the A0 model. 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, particularly between UML and SaveCCM. The main idea is creating a UML profile to allow developing UML models with domain-specific semantics. Further, the model is transformed to SaveCCM for time analysis and testing. 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. A characteristic of the approach is keeping up with standards using technologies such as UML, XML, XMI, XSLT and OCL. 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.

8. ACKNOWLEDGEMENT

This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS, and the Unity Through Knowledge Fund supported by Croatian Government and the World Bank via the DICES project.

9. 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] Akerholm, M., Moller, A., Hansson, H., and Nolin, M. 2005. Towards a Dependable Component Technology for Embedded System Applications. In *Proceedings of the 10th IEEE international Workshop on Object-Oriented Real-Time Dependable Systems* (February 02 - 04, 2005). WORDS. IEEE Computer Society, Washington, DC, 320-328. DOI= http://dx.doi.org/10.1109/WORDS.2005.52

[4] Crnkovic, I., and Larsson, M. 2002 *Building Reliable Component-Based Software Systems*. Artech House Publishers, 2002, ISBN 1-58053-327-2

[5] Hansson, H., Akerholm, M., Crnkovic, I., and Torngren, M. 2004. SaveCCM - A Component Model for Safety-Critical Real-Time Systems. In *Proceedings of the 30th EUROMICRO Conference* (August 31 - September 03, 2004). EUROMICRO. IEEE Computer Society, Washington, DC, 627-635. DOI= http://dx.doi.org/10.1109/EUROMICRO.2004.72

[6] IBM Rational Software Modeller web page: http://www.ibm.com/software/awdtools/modeler/swmodeler/, April 2008

[7] 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.

[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] Monot, A., and Noyrit, F. 2007. SaveIDE – Developer Documentation, October, 2007.

[10] OMG 2006. Object Constraint Language Specification. Version 2.0, May, 2006., http://www.omg.org/technology/documents/formal/ocl.htm

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

[12] Polak, M. 2005. UML 2.0 Components, Master Thesis, advisor: Vladimir Mencl, Charles Univ., Prague, September, 2005.

[13] Selic, B. 2007. A Systematic Approach to Domain-Specific Language Design Using UML. 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2007

[14] Sentilles, S., Hakansson, J., Pettersson, P., and Crnkovic, I. 2008. Save-IDE – An Integrated development environment for building predictable component-based embedded systems. Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), L'Aquila, Italy, September, 2008.

[15] 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.

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