Moazzam Fareed Niazi | Khalid Latif |
Tiberiu Seceleanu | Hannu Tenhunen

A Domain Specific Language for the SegBus Platform

Turku Centre *for* Computer Science

TUCS Technical Report
No 941, April 2009

A Domain Specific Language for the SegBus Platform

Moazzam Fareed Niazi
University of Turku, Department of Information Technology
Joukahaisenkatu 3-5 B, FIN-20520 Turku, Finland
moazzam.niazi@utu.fi

Khalid Latif
University of Turku, Department of Information Technology
Joukahaisenkatu 3-5 B, FIN-20520 Turku, Finland
khalat@utu.fi

Tiberiu Seceleanu
ABB Corporate Research
Västerås, Sweden
tiberiu.seceleanu@se.abb.com

Hannu Tenhunen
University of Turku, Department of Information Technology
Joukahaisenkatu 3-5 B, FIN-20520 Turku, Finland
hannu.tenhunen@utu.fi

TUCS Technical Report
No 941, April 2009

Abstract

The report presents a Domain Specific Language (DSL) for a multi-core segmented bus platform, *SegBus*. The DSL is based on Unified Modeling Language (UML) profile consists of graphical platform elements in the form of stereotypes with necessary tagged values to depict platform aspects at high level of abstraction. Customizations are applied to each stereotyped element in the form of user-defined rules to restrict relationship between platform elements. Object Constraint Language (OCL) is employed to introduce constraints to impose structural requirements between platform elements and also introduced mechanism to validate them. We present a simplified example of H.264 video encoder application where the DSL is used to specify and validate application and platform model in a unified representation manner. The DSL allow designers to model application on to platform in a correct and fast manner to get targeted performance and provides a key starting point for code generation that can be used in later stages of development cycle.

Keywords: Domain Specific Language, UML, SegBus

TUCS Laboratory
Distributed Systems Design

1 Introduction

With the continuous increase in digital system complexity, supported further by always decreasing technological figures, the platform based design perspective [1] does provide the means to address the challenge. The approach gains even more importance in the context of on-chip distributed or multiprocessing architectures.

Design decisions localized at higher system abstraction levels are known to bear the most impact on the quality of the eventual system implementation. On the other hand, optimality of design is strictly connected to platform parameters; hence, such platform level aspects, if taken into account at high abstraction levels will support a solution that takes full benefits from the features exposed by the platform. The specific platform we consider in this study is the *SegBus* platform [2].

The *Unified Modeling Language* (UML) [3] has been utilized in novel design methods proposing a solution for the challenge. We continue here the work towards establishing a full functional profile for the *SegBus*. The purpose is to provide a unitary framework for platform modeling, application mapping and system (platform+application) emulation, such that performance aspects are targeted, estimated and adjusted to optimal levels in a correct and fast manner. While the main aspects of the *SegBus* profile have been delivered by Lindroth et. al. [4], we address here issues related to the correctness (not formal) of the platform construction. Therefore, we continue here with an extension that imposes the employment of constraints, written in *Object Constraint Language* (OCL) [5]. This will further support a component based design approach, where the constraints are the platform specific topology and communication structure (elements that support the communication protocol on the platform).

The approach is based on building a *Domain Specific Language* (DSL), a specification language that is created to solve problems in a particular "domain" - the *SegBus* platform in our case. The proposed DSL enables us to model the platform with all necessary constructs and mapping of application components at certain abstraction level. Thus, we make the task of modeling application and platform fast and error-free by testing different platform configurations, and set the context for the development of a system emulator, for early stage performance assessment.

We thus build a graphical interface for the analysis of various *SegBus* instances that may answer, better or worse, to specific application requirements. The customization for each platform instance is defined in the form of user-defined rules. These customization rules set properties on each profile element about their relationships, ownerships, etc. The customization rules impose restrictions on profile elements during application / platform modeling, in order to provide a structurally correct version of the platform instance.

Related work. Defining a new language for any particular design flow is often considered to be a difficult task. With the evolution of platform-based design flow, it becomes necessary to build specification languages for each plat-

form that can represent platform concepts at high-level specification only. Many domain-specific languages have been built to encapsulate design and implementation knowledge from a particular application or technical domain.

Risi et al. [9] introduced *HyCom* - a DSL for hypermedia application development, particularly for describing hypermedia documents in a very declarative way. The DSL was embed in Haskell and hypermedia application can be constructed by combining and transforming domain components.

Consel et al. [10] introduced *Spidle* - DSL for for specifying streaming applications. A compiler also built for DSL to generate source code. The approach has been validated experimentally by comparing source code generated by Spidle compiler with equivalent C source code. A number of optimizations in Spidle compiler were missing like locality in data and instruction caches, performance impact of buffering input streat etc.

Arora et al. [11] presented a DSL for introducing application-level *Checkpointing and Restart* (CaR) mechanism in legacy applications for dynamic and distributed environments. The idea is to make sequential and parallel legacy system to be fault-tolerant by introducing code for CaR mechanism in high-level specifications.

Riccobene et al. [12] presented a UML profile for SystemC and defined a language to specify, analyze, design, visualize the software and hardware artifacts in a SoC design flow that provides a modeling framework for systems in which high-level functional models can be refined down to an implementation language. The work concluded that still there is a need to develop appropriate mechanisms and tools to fully utilize UML-based profiles system development with automation support.

Overview of the report. In the rest of the report, we proceed as follows. In section 2 we provide a short description of the *SegBus* platform and its structural characteristics. Next, in section 3 we provide description of proposed DSL including all involved phases from profile development to introducing structural constraints. Furthermore, in section 4 we provide a modeling example of H.264 video encoder in the context of proposed DSL to show its significance, followed by conclusion of report in section 5.

2 Background

2.1 Segmented Bus Architecture

A segmented bus is a “collection” of individual buses (segments), interconnected with the use of FIFO like structures. Each segment acts as a normal bus between modules that are connected to it and operates in parallel with other segments. Neighboring segments can be dynamically connected to each other to establish a connection between modules located in different segments. Due to the segmen-

tation of the bus lines, and their relative isolation, parallel transactions can take place, thus increasing the performance. A high level block diagram of the segmented bus system which we consider in the following sections is illustrated in Figure 1.

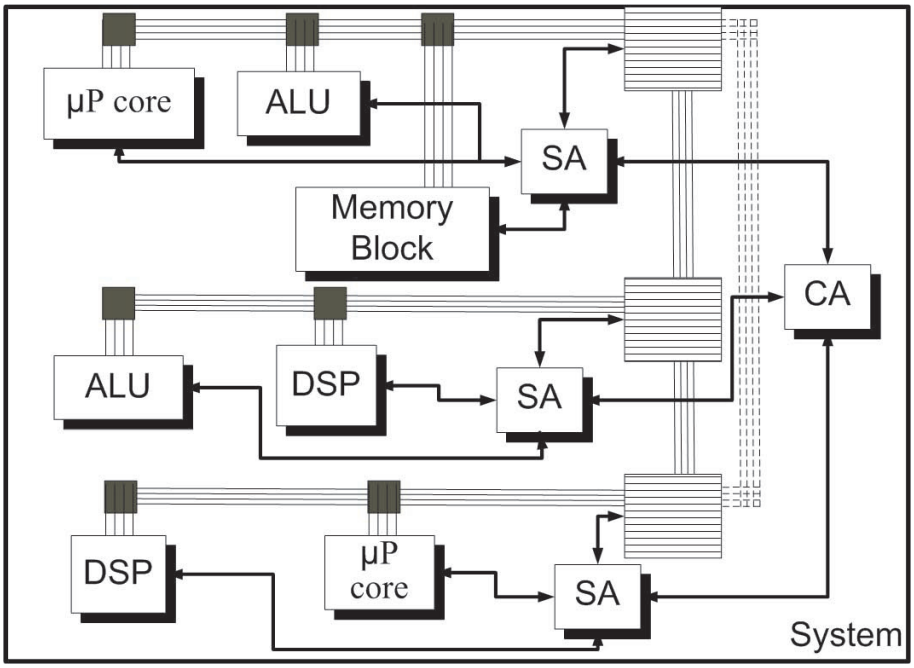

Figure 1: Segmented bus structure.

The *SegBus* communication platform is composed of components that provide the necessary separation of segments - *Border units* (***BU***), arbitration units - the *Central Arbiter* (***CA***) and local, *Segment Arbiters* (***SA***). The application then is realized with the support of (library available) *Functional Units* (***FU***).

The *SegBus* platform has a single ***CA*** unit and several ***SA***s, one for each segment. The ***SA*** of each bus segment decides which device (***FU***), within the segment, will get access to the bus in the following transfer burst.

Platform communication. Within a segment, data transfers follow a "traditional" package based bus protocol, with ***SA***s arbitrating the access to local resources. The inter-segment communication, is also a package based, circuit switched approach, with the ***CA*** having the central role. The interface components between adjacent segments, the ***BU***s, are basically FIFO elements with some additional logic, controlled by the ***CA*** and the neighboring ***SA***s. A brief description of the communication is given as follows.

Whenever one ***SA*** recognizes that a request for data transfer targets a module outside its own segment, it forwards the request to the ***CA***. The later identifies the target segment address and decides which segments need to be dynamically connected in order to establish a link between the initiating and targeted devices.

When this connection is ready, the initiating device is granted the bus access, and it starts filling the buffer of the appropriate bridge with the package data. Following a signaling protocol, the data is taken into account by the corresponding next segment ***SA***, which forwards it further, towards the destination. At this point, the ***SA*** of the targeted segment routes the package to the own segment lines, from where it is collected by the targeted device.

A transfer from the initiating segment k to the target segment n is represented in Figure 2. The segments from k to n are released for possible other inter-segment operations in a cascaded manner, from the source k to the destination, n.

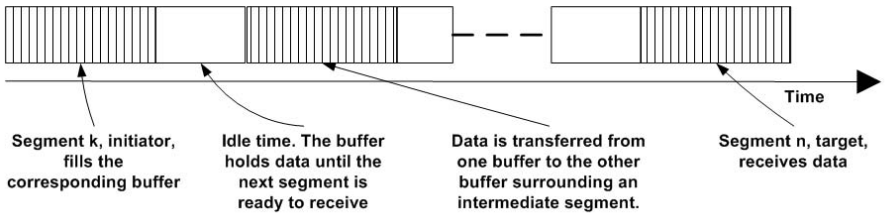

Figure 2: Inter-segment package transfer.

The arbitration at ***CA*** level implements the application dataflow, with respect to these transfers. Henre, one has to implement accurate control procedures for inter-segment transfers, as possible conflicting requests must be appropriately satisfied, in order to reach performance requirements and to correctly implement applications.

2.2 The *SegBus* UML Profile

Lindroth et. al [4] provided the initial steps of the *SegBus* platform profile. It contains a hierarchical decomposition of platform components and provides appropriate means for characterization, instantiation and connectivity but some of the important features were missing, such as model validation according to platform definition, attributes of platform elements, structural constraints etc. Figure 3 shows the profile elements.

The profile contains the structural elements of the platform. It contains the platform itself, the stereotype *SegBusPlatform*, one element modeling the segments, *Segment*, the stereotype representing the ***SA***, *SegmentArbiter*, stereotype *CentralArbiter* represents ***CA***, etc. A *metaclass* is a class whose instances are classes. Here, all the elements are generalization of metaclass *uml20.classes.Class*.

2.3 Platform Constraints

The profile described above provides us with the environment for a UML based platform specification and application development. However, further elements

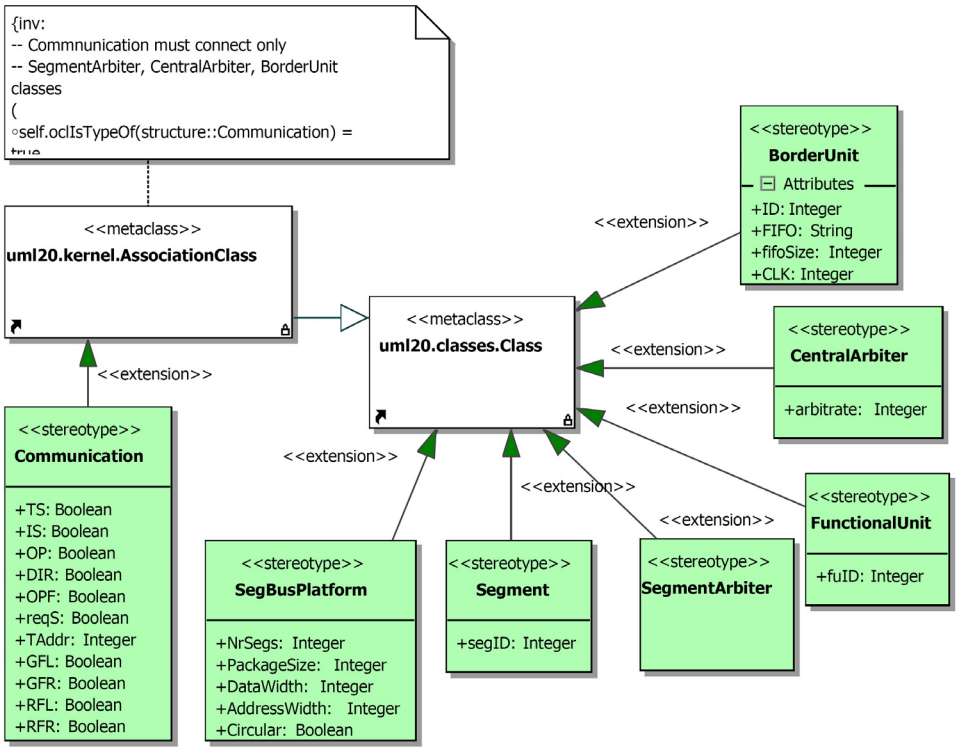

Figure 3: The SegBus profile elements [4].

must be taken into consideration, such that a specific platform instance is not built in an incorrect fashion. For this, the characteristics of the *SegBus* must be considered. A few of these constraints are described as follows.

- The platform may have either a linear or a circular geometry. The topology impacts on how the "terminal" segments are connected to each other.
- Every platform instance has a unique ***CA***.
- Every segment has a unique ***SA***.
- Every ***SA*** is connected to at most two ***BU***s.
- Every ***BU*** is connected to at most two ***SA***s.
- Every segment contains at least one ***FU***.

In the following, we express these platform characteristics as structural constraints and connect them to the *SegBus* UML profile, such that a correct component approach to platform design is implemented.

3 DSL for the SegBus Platform

Domain-specific modeling (DSM) is a way of designing systems that involves the systematic use of domain-specific languages (DSLs) to represent the various facets of a system. DSL tend to provide higher-level abstractions than general-purpose modeling languages like UML.

DSL encapsulates domain concepts and provide semantics to domain entities, allowing designers to aware themselves as working directly with domain concepts. DSL is build when there is a good understanding of the problem domain. We employ *MagicDraw UML* [6] tool to graphically model various artifacts of the proposed DSL, as the tool not only provides UML capabilities, but also provides *DSL Customization Engine* - an engine able to process user-defined rules for DSL elements and reflect this in graphical interface and diagrams behavior.

Figure 4 provides a general overview of the proposed DSL. At the top level, we transform platform concepts into the high-level graphical constructs to form a DSL, specific for the *SegBus* platform. The DSL provides a graphical environment where a designer can map *Platform Independent Model* (PIM) of the application on to platform quickly and assign pre-existing components from the *SegBus Component Library* during modeling. Finally, the model can be validated for possible mistakes to get a correct *Platform Specific Model*.

We have developed the "*SegBus* DSL" over three main directions: *Profile De-*

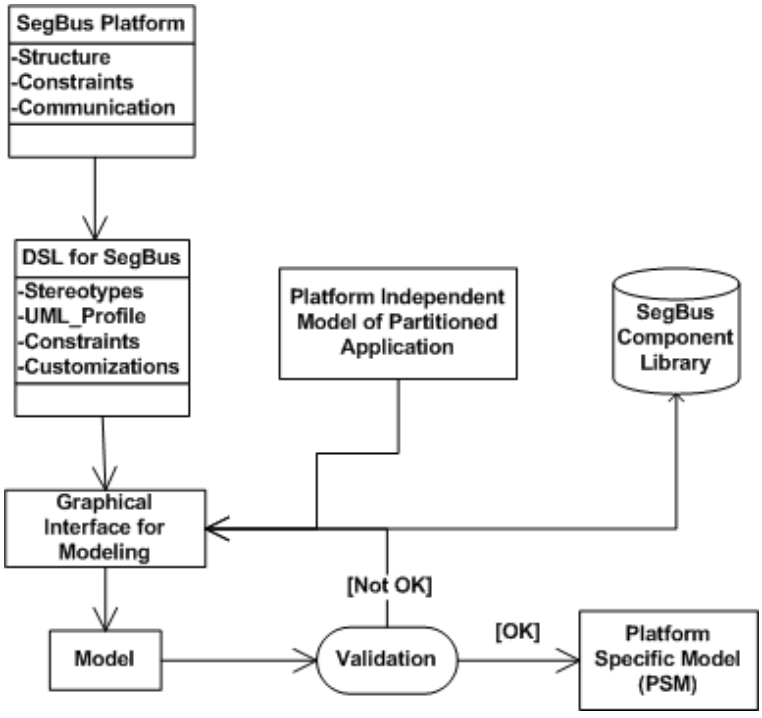

Figure 4: Design process of the *SegBus* DSL.

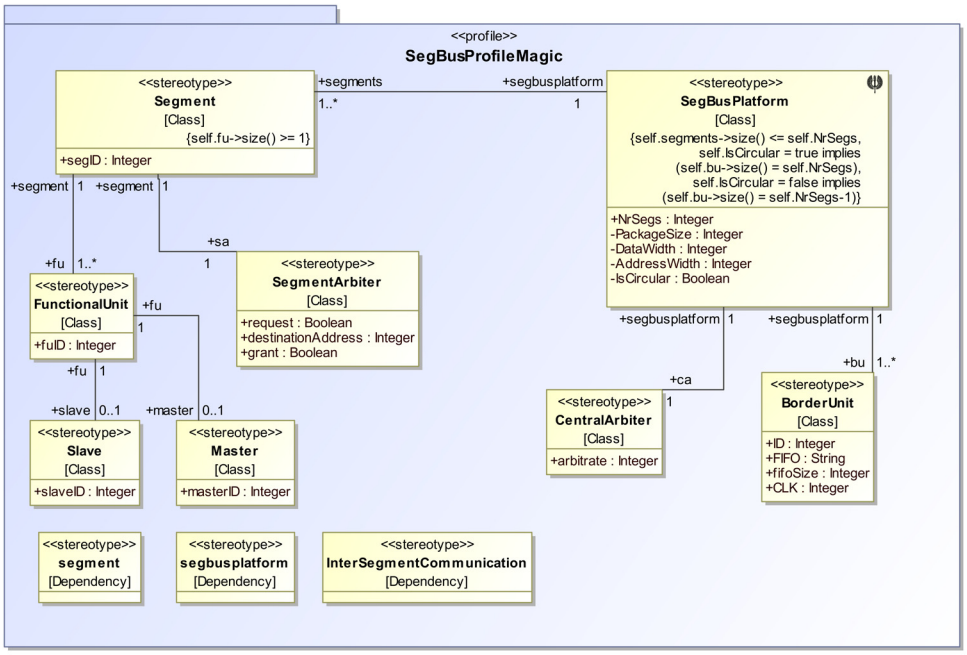

Figure 5: Platform elements and their association in profile.

velopment, *DSL Customization*, *Structural Constraints*. We briefly discuss related aspects in the next sub-sections.

3.1 Profile Development

The structural characteristics of the platform are the key starting point of profile development. These are analyzed to develop the UML elements to depict the hardware components of the platform in high level models. The modeling of the hardware resources of the *SegBus* platform in UML profile starts by analyzing the platform itself. A UML package *SegBusProfileMagic*, with stereotype *profile* is a collection of classes with stereotypes to sustain application development on the *SegBus* platform. The profile defines the main structural elements of platform. All the classes in the profile that model a particular element of platform are generalizations of the metaclass *UML Standard Profile::UML2 Metamodel::Classes::Kernel::Class*. The structural view of the profile is depicted in Figure 5 with necessary association and multiplicities between profile elements.

Figure 6 shows the hierarchical structure of the platform elements. At the top level is the *SegBusPlatform* itself composed of *Segment*(s) and exactly one *CentralArbiter*. Every *Segment* is be composed of several *FunctionalUnit*s, which may be a *Master* or *Slave* as per application requirement, and exactly one *SegmentArbiter*. Each *Segment* is be connected with other *Segment* through *BorderUnit*s.

The platform (*SegBusPlatform*) is characterized by the number of segments

it contains, platform geometry (linear/circular), package size for communication, data width and address lines. Each *Segment* is composed of one *SegmentArbiter*, more than one *FunctionalUnit*s (master and/or slave) and interfaces to the neighboring segments via *BorderUnit*. The important and central element of each segment is the *SegmentArbiter* that use to coordinates both the intra-segment and inter-segment communication. The units involve in writing data on to the bus are called *active* units. These units are represented by the *Master*s and they're contained by *FunctionalUnit*. One *FunctionalUnit* may contain up to one *Master* and one *Slave* as depicted in Figure 5. The *FunctionalUnit*'s ID (natural number that is unique at system-level) is inherited by both contained *Master* and *Slave*. The *FunctionalUnit* methods contain procedures to produce data and to communicate with *SegmentArbiter*. Procedures for sending and receiving data are placed within *Master* and *Slave* respectively.

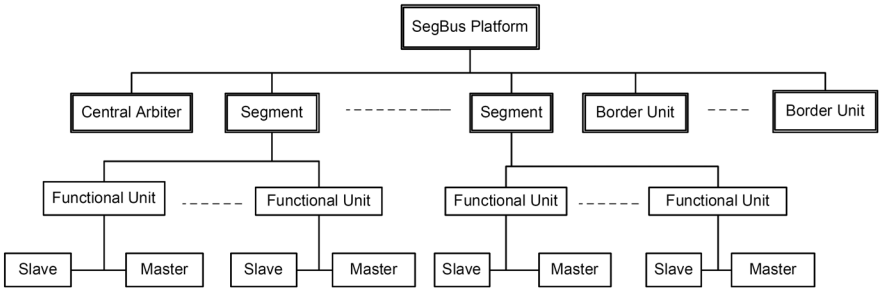

Figure 6: Hierarchical structure of the SegBus profile elements.

The *BorderUnit* element is an interface between one segment and its neighbours. The internal FIFO buffer is characterized by the *fifoSize* tag.

3.2 DSL Customization

The next step in DSL development is to introduce user-defined rules for each profile element. All user-defined rules for each profile element are stored in customization classes. Customization classes are generalization of metaclass *UML Standard Profile::MagicDraw Profile::DSL Customization::Customization* class, with stereotype *Customization*. These customization classes comprise of tags that store the user-defined DSL customization rules. The customization rules are parsed and interpreted by the *DSL Customization Engine* to assist validation process. A UML package is created to store all customization classes.

Figure 7 shows the user-defined rules for each profile element. We illustrate the usage of a few customization rules as follows.

- **customizationTarget.** This tag stores the names of stereotype(s) which we are going to customize with respect to current class. User-defined rules which we introduce in the customization class will be applied to all stereotypes classes that are mentioned in this tag. In Figure 7, the customization

customizationTarget	possibleOwners	inShortCutMenu	suggestedOwnedTypes
SegBusPlatform	Package	NrSegs,PackageSize,DataWidth, AddressWidth,IsCircular	Segment,CentralArbiter,BorderUnit
Segment	SegBusPlatform	segID	SegmentArbiter, FunctionalUnit
SegmentArbiter	Segment	-	-
CentralArbiter	SegBusPlatform	-	-
BorderUnit	SegBusPlatform	ID, fifoSize	-
FunctionalUnit	Segment	fuID	Master,Slave
Master	FunctionalUnit	masterID	-
Slave	FunctionalUnit	slaveID	-

Figure 7: User-defined rules for different attributes of the Customization classes.

targets are in the first column i.e. *SegBusPlatform, Segment, SegmentArbiter, CentralArbiter, BorderUnit, FunctionalUnit, Master* and *Slave* stereotyped elements.

- **possibleOwners.** This tag consists of stereotyped or other UML elements that can instantiate current element. As of the second row of Figure 7, the possible owner of *Segment* can only be *SegBusPlatform*, which can instantiate it inside the respective class.

- **inShortCutMenu.** This tag is used to add attributes of a class in shortcut menu. The value of these attributes can be set by right-clicking on any specific model element. In first row of Figure 7, we included *NrSegs, PackageSize, DataWidth, AddressWidth, IsCircular* properties, which appear as shortcut menu items in the context of the *SegBusPlatform* class.

- **suggestedOwnedTypes.** This tag contains list of stereotypes and metaclasses whose object can be instantiated inside the stereotyped class as inner elements. In first row of Figure 7, *SegBusPlatform* can only be associated with *Segment, CentralArbiter* and *BorderUnit* stereotyped classes.

We've also introduced three different customized *Dependency* links (Figure 5), in order to connect different stereotyped elements of the *SegBus* platform according to needs. The advantage of customizing such links during DSL development is to specify what will be the possible source and target stereotype(s) for given links. The customization of these links allow designer to connect only particular platform elements by imposing user-defined rules. In Figure 8, the two elements *Segment* and *BorderUnit* are connected with a customized link *InterSegmentCommunication*. The link imposes specific properties of the platform for communication between mentioned platform elements.

In Figure 9, we depict the customization classes of the *SegBus* DSL.

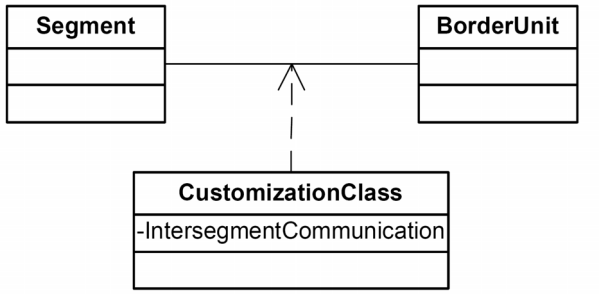

Figure 8: Dependency link between two profile elements.

Figure 9: DSL Customization classes for each element of the *SegBus* platform.

3.3 Structural Constraints

The platform characteristics defined in section 2.3 need to be introduced in the form of structural constraints in the DSL. We specified the required constraints by using the *Object Constraint Language* (OCL v2.0) [5] and relate them to the *SegBus* UML profile, such that a correct component approach to platform design is implemented. Some of the constraints are already introduced when applying multiplicities to relationships between elements in profile the development phase. For instance, there should be exactly one *CentralArbiter* in whole *SegBusPlatform* - modeled by specifying multiplicity as '1' (Figure 5). Also, it is important to

enforce in design that the number of *Segment*s should be equal to the property *NrSegs* of *SegBusPlatform*, number of *BorderUnit*s should match the platform geometry, etc.

All the constraints are stereotyped as *validationRule* from the *Validation Profile*, a profile supplied by the tool for supporting the validation of models. The *Validation suite* defines a set of validation rules, to be applied when validating a model. The purpose of making a validation suit is to group constraints logically in a UML package, *SegBus Constraints*, stereotype with *validationSuit* with proper context supplied for each constraint. We apply this validation suit on our models when validating it against the platform constraints.

Upon any breach of any constraint requirement during the design process, the tool provides an error message with a text specified by the DSL. The designer can subsequently try to solve the indicated problem. The description of a few of the constraints that we introduced in the DSL, and of the related messages are given below:

- **Number of Segments**: This constraint enforces the number of *Segment*s in the model to be equal to the value of the integer attribute *NrSegs* that we specified in the stereotype *SegBusPlatform*. *NrSegs* represents the number of segments that we required in the platform. The constraint specification is given as:

  ```
  context SegBusPlatform
  inv NrOfSegments:

  self.segments->size() = self.NrSegs
  self.segments->size() > 1
  ```

  Error Message 1. "Number of segments in model are not same as specified in SegBusPlatform".

  Cause. The designer introduced more / less segments than the specified number.

- **Number of FuncationalUnits in a Segment**: This constraint enforces in model that each segment must contain at least one *FunctionalUnit*.

  ```
  context Segment
  inv NumberOfFU:

  self.fu->size() >= 1
  ```

  Error Message 2. "There should be more than one Functional Unit in each segment"

  Cause. A segment does not contain any *FunctionalUnit*.

- **Number of BorderUnits**: This constraint enforces in design that there must be a number of *BorderUnit*s matching to number of segments with respect to platform geometry.

```
context SegBusPlatform
inv NumberOfBorderUnits:

if self.IsCircular = true then
    self.bu->size() = self.NrSegs
else
    self.bu->size() = self.NrSegs-1
endif
```

Error Message 3. "Number of Border Units is not compliant given the selected platform topology".

Cause. The wrong number of border units has been included in the design.

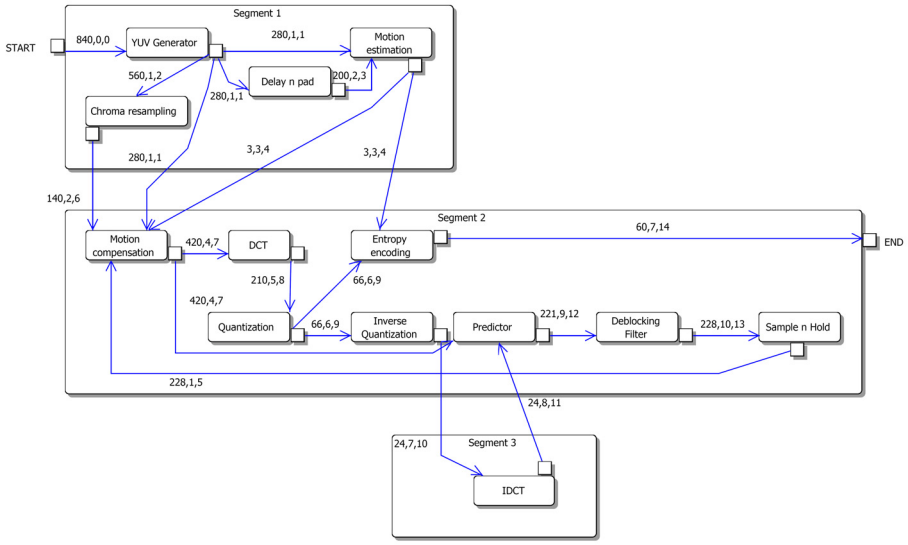

Figure 10: H.264 video encoder partitioned application model.

4 Example using Modeling Tool

We demonstrate our approach with an example of modeling the H.264 video encoder on the *SegBus* platform, using the developed DSL.

Following the previous work [8], we have already decided on a platform structure: three segments, linear topology. The application has already been partitioned

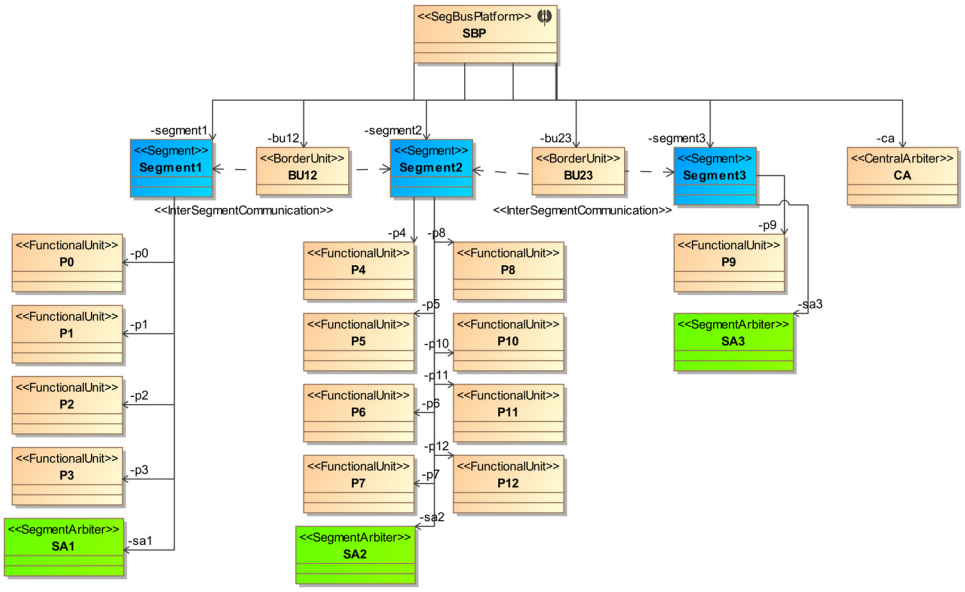

Figure 11: Example configuration of the *SegBus* platform with 3 segments and linear topology.

for this case, as depicted in Figure 10. We specify this information in an instance of the *SegBusPlatform* class's. The respective attributes are: $NrSegs = 3$, $IsCircular = false$ and also set other information, such as address and data bus width, etc. Finally we map components of the partitioned application model on the particular segments. The configuration can be seen in Figure 11.

In order to check the structural correctness of our design, we run the validation suite *SegBus Constraint*. This can be executed at any time during the platform development process, that is, after any change has been performed, such as the introduction of a new functional unit, moving one functional unit into another segment, etc. If the validation action is performed, for instance, immediately after the instantiation of the segments, the tool will provide us with three error messages (nr. 2), and a highlighting of the three segment instances, as we did not yet associate with them any functional units. By taking the corresponding actions, these error messages will disappear.

As another example, suppose that instead of three segments, we introduce a fourth one. This will conflict with the "number of Segments" constraint, and the tool will provide to us the error message nr. 1.

5 Conclusions and Future Work

The report presented methods for specifying, modeling and implementing multi-core embedded system using UML-based methodology. We introduced a DSL

for modeling and mapping of segmented bus architecture - *SegBus* with desired application. We described in the form of graphical elements the principal structural elements of the platform with their structural relations and the related DSL customization.

The DSL provides an environment where a designer can model platform and associate it with application components in a fast manner using different configurations. It will help designers to correctly model application and platform in a fast manner and will help in model transformation at later stages of development process. The DSL doesn't allow modeling the platform by violating structural constraints. The validation suit embedded in DSL helps designer to rectify the problems in model and correct them with necessary measures. In subsequent steps of the design process, we can also generate code from the model and able to analyze its efficiency in the emulation program, so that proper adjustment could be done in models to achieve optimal performance from platform.

The presented DSL will be used for code generation for any modeled *SegBus* configuration and an emulation program needs to be developed for early performance estimation. The emulation program will help to optimize high-level models of the platform to achieve maximum performance from the platform

References

[1] Ferrari, A., Sangiovanni-Vincentelli, A. *System design: Traditional concepts and new paradigms.* In Proceedings of IEEE International Conference on Computer Design: VLSI in Computer and Processors, pp. 212, 1999.

[2] T. Seceleanu. *The SegBus Platform - Architecture and Communication Mechanisms.* Journal of Systems Architecture (2006), doi:10.1016/j.sysarc.2006.07.002

[3] www.omg.org. *UML Superstructure Specification, v2.0.*

[4] T. Lindroth, R. Lavinia, T. Seceleanu, N. Avessta, J. Teuhola. *Building a UML Profile for On-chip Distributed Platforms.* In Proceedings of the 30th Annual International Computer Software and Applications Conference, 2006

[5] OMG. *OCL 2.0 Revised Submission, version 1.6* January 6, 2003.

[6] MagicDraw UML. http://www.magicdraw.com

[7] Model-Driven Architecture. http://www.omg.org/mda/

[8] K. Latif, M. Niazi, H. Tenhunen, S. Sezer, T. Seceleanu. *Application development flow for on-chip distributed architectures.* In proceedings of 21st IEEE International SOC Conference (SOCC), pp. 163-168, 2008.

[9] W. Risi, P. López, D. Marcos. *HyCom: A Domain Specific Language for Hypermedia Application Development.* In proceedings of 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 9, pp. 163-168, 2001.

[10] C. Consel, H. Hamdi, L. Réveillére, L. Singaravelu, H. Yu, C. Pu. *Spidle: a DSL approach to specifying streaming applications.* In Proceedings of Proceedings of the 2nd international conference on Generative programming and component engineering, pp. 1-17, 2003.

[11] R. Arora, M. Mernik, P. Bangalore, S. Roychoudhury, S. Mukkai. *A Domain-Specific Language for Application-Level Checkpointing.* In Proceedings of International Conference on Distributed Computing and Internet Technologies (ICDCIT 2008), pp. 26-38, 2008.

[12] E. Riccobene, A. Rosti, P. Scandurra. *Improving SoC Design Flow by means of MDA and UML Profiles.* In 3rd Workshop in Software Model Engineering (WiSME), 2004.

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5B, FIN-20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 978-952-12-2286-3
ISSN 1239-1891