Domain-specific front-end for virtual system modeling

Janne Vatjus-Anttila, Jari Kreku, Kari Tiensyrjä

VTT, Technical Research Centre of Finland, Oulu Finland
{janne.vatjus-anttila, jari.kreku, kari.tiensyrja}@vtt.fi

Abstract. The complexity of software and hardware in embedded systems has risen rapidly due to convergence of diverse applications and adoption of multi-core technologies. Consequently, the abstraction level of system design, modeling and exploration needs to be raised to manage the complexity. The Y-chart approach, typically applied in the system-level performance evaluation, allocates/maps a model of application on a model of execution platform and the resulting system model is simulated to obtain performance data. In this work, Domain-Specific Modeling (DSM) has been adopted as means of raising the abstraction level for building, composing, configuring and checking of high-level models in the virtual system performance modeling and simulation approach, called ABSOLUT. Domain-Specific Languages (DSL) were defined to serve as front-ends for application workload, platform and allocation modeling using the MetaEdit+ tool. The results are demonstrated with a video player case example. First experiences indicate that in performance evaluation related tasks the modeling productivity, model management and ease of learning have improved.

Keywords: DSM, DSL, embedded system, virtual system, SystemC, performance exploration

1 Introduction

The complexity of hardware and software has risen rapidly particularly in advanced embedded system domains, like communication systems, during the recent years due to extensive adoption of multi-core technologies. Real-time embedded systems are often computationally intensive and constrained with limited resources, e.g. power/energy, size and cost. The systems accommodate a large number of on terminal and/or downloadable applications offering the users with numerous services related to telecommunication, video, digital television, internet etc. More flexibility, scalability and modularity are expected from the execution platforms to support the applications. The digital processing architectures will evolve from current system-on-chips to massively parallel computers consisting of heterogeneous subsystems connected by a network-on-chip.

The design complexity requires the elevation of the design process to a higher level of abstraction. At the system level, models of entire platforms can be built that enable hardware-software co-development and rapid, early design space exploration. Such models provide quick feedback for the designers about the effect of their design decisions to critical system metrics like performance. Complex interactions and the highly dynamic nature of systems make their static analysis difficult, which is why such executable models are indispensable [1].

In this work we use ABSOLUT [2] methodology and toolset as a backend for early phase system level performance simulation of embedded systems. The main modeling phases in the virtual system modeling include specifying computing platform’s capacity model, application workload and allocation of workload to computing resources of the platform. The result of an allocation is a virtual system model, which can be simulated using the OSCI SystemC simulator to measure performance data, e.g. utilization of platform resources.

Domain-Specific Modeling (DSM) raises the level of abstraction beyond programming by specifying the solution directly using domain concepts. It is used in many application domains and in particular embedded application development with domain specific language (DSL) has been adopted in many companies [3].The final products are generated from these high-level specifications [4]. The benefits of DSM come in many forms like easier modeling/programming, productivity increase, better code quality and maintenance ability [5].

In this paper we propose applying DSM in virtual system modeling domain and present a MetaEdit+[6] based prototype DSL that can be used as a front-end to the ABSOLUT performance modeling and simulation approach. It resembles a traditional Graphical User Interface (GUI) of modeling tool and it can be used like one. We applied the DSL in an example case study and present the enhanced performance evaluation workflow using DSM.

The rest of the paper is structured as follows. In Chapter 2 system-level performance modeling and evaluation and used ABSOLUT approach are described. Chapter 3 discusses Domain-Specific Modeling. Chapter 4 presents the DSL front-end for ABSOLUT approach through applying the DSM method for virtual system model development phases. In Chapter 5 the ABSOLUT workflow and the use of the DSL in it is presented and results of the case study are shown. Chapter 6 gives conclusions.

2 System-level performance modeling and evaluation

Performance evaluation approaches can be divided into three categories: analytical approach, simulations and measurements [7]. The analytical approach is suitable for early performance evaluation, but the accuracy of results is low because it requires many simplifications and assumptions. In simulations, the execution of an application is simulated using a computer program. Simulation provides more accurate results than analysis since it is possible to incorporate more details of the system in the models. Simulations are suitable for early evaluation, since they can be performed before implementations of hardware and/or software are completed. Measurements can be done with real applications, prototypes of applications or benchmark programs that mimic real software. An implementation of the execution platform is, however, required in all cases and therefore measurements are not suitable for early evaluation.

Performance simulation approaches are categorized in the European EDA Roadmap [8] into virtual systems, virtual platforms and virtual prototypes:

- Virtual system approaches combine abstract application models with an abstract execution platform model. The applications are represented using e.g. workload models, traces or task graphs but not as real instructions of processors. The platform model typically has a high abstraction level and capacity models of components instead of instruction set simulators.
- Virtual platform approaches use real application software compiled to binary form in simulations. The execution of the applications is simulated on top of a virtual platform model, which contains one or more instruction set simulators. The platform models need to be functionally complete and use accurate memory maps to be able to execute the application binaries.
- Virtual prototype approaches also use real application binaries with instruction set simulators. The virtual prototype approaches model full functionality of the execution platform using hardware description languages like VHDL or Verilog.

ABSOLUT [2] is a virtual system performance simulation approach intended for early evaluation of embedded computer systems and for exploring the design space. It is also a set of tools, which assist the designer with application and platform modeling, allocation and configuration, simulation and result visualization. It has been applied to several case studies ranging from contemporary mobile phone platforms to future high-performance systems consisting of hundreds of components. Fig. 1 presents the main phases of ABSOLUT performance modeling method.

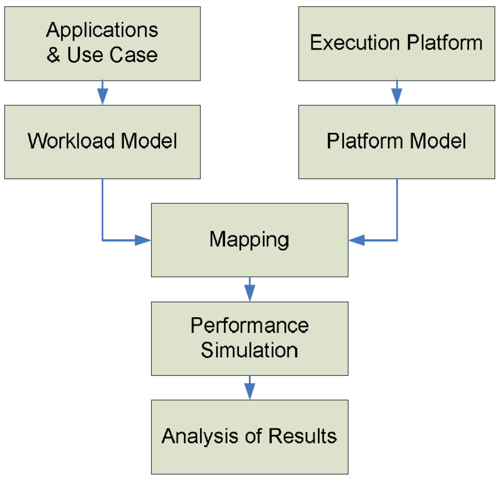


Fig. 1. Y-chart model of ABSOLUT performance modeling

Applications are modeled as layered workload models, which ultimately consist of abstract, instruction-like workload primitives. Several techniques have been developed to create workload models from information sources such as application specifications or execution traces. A compiler-based tool exists to create workload models automatically from application source code [9].

As application functionality is abstracted, the complexity of execution platform models is also reduced especially with respect to the processing elements. The data paths of processing elements need not be modeled in detail and data transfers and storage are simulated only from the performance point of view. A capacity model of platform can be rapidly constructed from components in a model library with the help of a platform generation tool.

The virtual system model, constructed by allocating workload models on top of the platform components, is simulated using the IEEE standard SystemC [10] simulation kernel and models based on the TLM standard [11]. Performance, power and energy consumption is obtained from simulation by instrumenting the workload and/or platform models with custom performance probes. Designer can freely set the probes and extract e.g. resource utilization, execution latencies or interconnection traffic.

3 Domain-Specific Modeling

Basically, Domain-Specific Modeling is creating and using modeling languages for specific purposes. Domain-specificity of modeling means that key concepts of each domain can be used as the modeling elements of the language. Modeling elements can have graphical symbols and the use of DSL is actually placing these elements on diagram according language modeling rules. Diagrams made with DSL are formal descriptions of systems and applications and as such suitable for documentation, analysis and transforming them to other forms like source code or other artifacts. In particular, the generation features complete the benefits that DSM and DSLs provide.

Tool support for applying DSM exists in commercial, academic and open source tools [12]. True DSM tools enable developing of new DSL, which can be run on top of the tool or as a standalone program depending of the tool. Increasing interest towards DSM has brought DSM features also to IDEs [13].

DSLs are often made to very specific domains inside companies, which are not in public use. However, DSLs are applied in the graphical user interfaces of some embedded system development tools, too. For example, CoFluent Studio [14] and Simics [15] include DSL that can be used for modeling of explored system. These apply for somewhat similar purposes as ABSOLUT and have therefore similar modeling phases. However, the modeling notions differ due methodology differences. In addition, it is important to notice the fundamental difference between the ABSOLUT DSL front-end and the DSLs of mentioned tools. Neither GUIs nor DSLs of the mentioned tools are implemented on an actual DSM tool as the front-end presented in this paper is.

4 DSL front-end for virtual system modeling

Virtual systems for performance simulations are often made from existing components and in an ideal case, the modeling should not require much coding. However, the platform model needs to be composed and configured. The application modeling requires making a high-level model of application. The allocation of application elements to the processing elements of platform model is one of the modeling phases.

In this work, the DSM approach is applied to the modeling phases. The same DSM environment that consists of a DSM tool, a few DSLs and ABSOLUT tools applies also for building simulateable models, running simulations and analyzing results.

4.1 DSM workflow and environment

Although every domain has its special features, a workflow consisting of the next four phases for developing DSLs for different domains can be used [16]:

1. Identifying abstractions and how they work together
2. Specifying the language concepts and their rules (meta-model)
3. Creating the visual representation of the language (notation)
4. Defining the generators for model checking, code, documentation, etc.

In our case, the Y-chart model shows clearly the modeling phases and their inter-relations. Consequently, it was natural to proceed towards modeling phase specific abstractions. The existing ABSOLUT model library and experiences of tools of the domain were the basis according which the language concepts, properties, rules and notations were defined. The notation design in prototype development phase had naturally low priority. In the generator definitions, the goal was to enable the usage of the existing ABSOLUT tools and generation of compatible mid- representations.

Proceeding according to the workflow requires expertise of both the problem domain and the DSM. Additionally an appropriate DSM tool is needed. In this work, the DSM tool MetaEdit+ 4.5 Workbench [6] developed by MetaCase has been used and the solution presented here contains some tool specific notions. The tool provides different diagram editors for modeling with the DSM languages, support for the DSL and generator definition and is upgraded with new versions by the tool vendor.

4.2 DSL for workload modeling

The workload modeling DSL is developed for the compiler based workload generator of ABSOLUT, which produces workload models from normal C/C++ code. Any application modeling DSL that can generate C code can therefore be used in producing the ABSOLUT compatible workload model. Existing C code is also used for workload generation. The workload modeling DSL of ABSOLUT front-end is targeted for workload model and workload trace configuration and generation.

The workload modeling is made with the diagram presentations of *Workload Modeling Graph*, on which the DSL objects and connections are placed. Fig. 2 presents an example of such a diagram. It contains two *Workload Model* objects, which both are linked to two *Workload Trace* objects. The object symbols contain symbol names and information about the status of the workload models and traces. The MetaEdit+ toolbar contains the modeling elements of the graph type and the generator buttons.

The workload modeling scheme consists of two phases and both have an own object notation in this solution. The *Workload Model* object is used for defining and generating the workload model according to the object properties. Properties are used to select the valid external workload model generator, the generation script and the source code folder. The MERL workload model generator uses the property information and generates the workload model. Several *Workload Model* objects can be used in a single diagram to generate different workload models from the same source code or from many different source codes.

The *Workload Trace* object is used to make the workload trace that can be allocated to the platform model. It needs a link to the *Workload Model* object defining the workload model that is executed in trace generation. The properties of the *Workload Trace* object define all the parameters that are needed to execute the workload model. The object has also a property, which defines path used to store the generated trace.

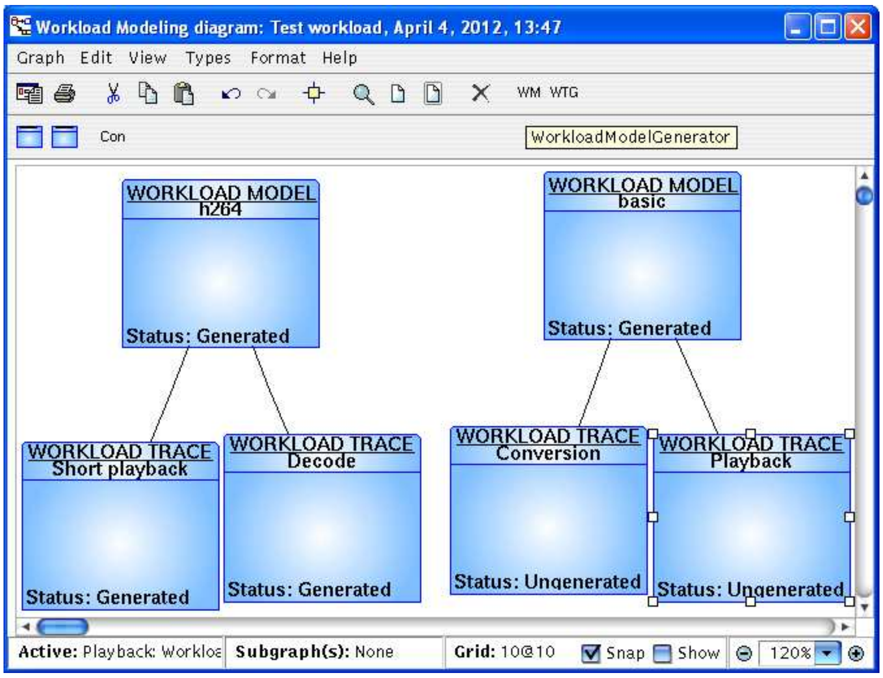


Fig. 2. Diagram presentation of *Workload Modeling Graph*

The MERL generator first runs the workload model that produces the workload trace and then stores the trace files to defined path. Generated workload trace consists of files of which each contains trace of one workload model thread. Several *Workload Trace* objects can be linked to single *Workload Model* object, which enables producing different workload traces from the workload model based on the parameters.

4.3 DSL for platform modeling

The key concepts that are needed in the platform modeling are the various hardware elements (see Table 1). The element types and their properties are specified with different property sets. In addition to the objects, some relationship types and role types are needed for linkage between hardware elements. If large platforms are modeled, the sub-system object could be applied in the DSL too. Modeling rules can be included in the DSL to prevent the designer from making impossible or erroneous connections between elements or other kind of modeling mistakes.

A diagram editor is the only alternative for a block based platform modeling. Modeling work convenience depends of the modeling elements. By using different symbols for the language concepts, perceiving of the platform model is easier. Different shapes, colors and text of the component objects symbols enable this. The relationship and role symbols can also have tuned symbols, if there are many of them or if they have properties.

Generators can have more than one purpose in the platform modeling DSL. Checkup generators are another way to confirm the platform validity in addition to the rules set in the DSL definition. The main purpose of the generator in the platform modeling is to produce a description of a platform that can be used in the simulation phase. Generators can also be used to update the modeling element list, which is important when the components are modeled elsewhere.

Our DSL for platform modeling consists of platform objects, their relationships, roles, modeling rules and the platform model to XML generator. The objects and other elements of the platform modeling DSL are listed in Table 1.

Table 1. Modeling elements of platform modeling DSL.

Element	**Description**
Processor	Object is used to model processor and accelerator components.
Memory	Object is used to model memory components.
Bus	Object is used to model bus components.
Interface	Object is used to model interface of subsystem.
Subsystem	Object is used to model subsystems of platform.
Router	Object is used to model connections between subsystems.
Connection	Relationship used to model all connections in the platform diagram.
Master	Role that connects master component to connection.
Slave	Role that connects slave component to connection.

The integration of the DSL with ABSOLUT and its component library is established by importing the component library description to the DSL. The import updates the pull down list of each hardware element. For example, a new processor type can be selected from the pull down list of *Processor* object types when an updated ABSOLUT component library definition has been imported.

An example ARM processor platform, modeled in a diagram presentation of the *Platform Modeling Graph* is presented in Fig. 3. Using of the platform modeling DSL resembles the use of many other platform-modeling tools. Components are picked from toolbar, placed, connected to other components and configured from the properties of the particular component type.

Subsystem objects are also defined in the DSL to help the modeling of large platforms in smaller parts. When they are used, the top level of platform is composed from *Subsystem* and *Router* objects. The architecture of each subsystem can be modeled with a separate platform diagram as Subsystem object decomposition. The i*nterface* objects are used inside them to define how they are connected to other subsystems.

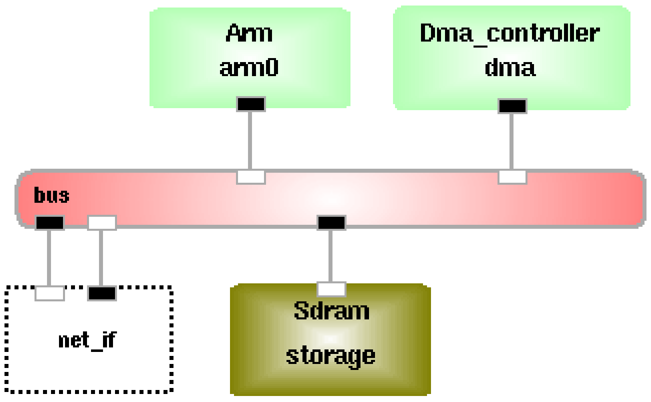


Fig. 3. Platform model made with platform modeling DSL

The MERL generator is used to generate XML descriptions of the modeled platforms. The generator goes through all the platform diagrams that are related to the platform model and includes their connections, components and the whole platform structure into the XML file.

4.4 DSL for allocation

An allocation DSL should contain objects presenting application model elements and computing resources of platform. Some way to link application model elements to selected platform model elements is also required. A generator for generating allocation file belongs also to allocation DSL.

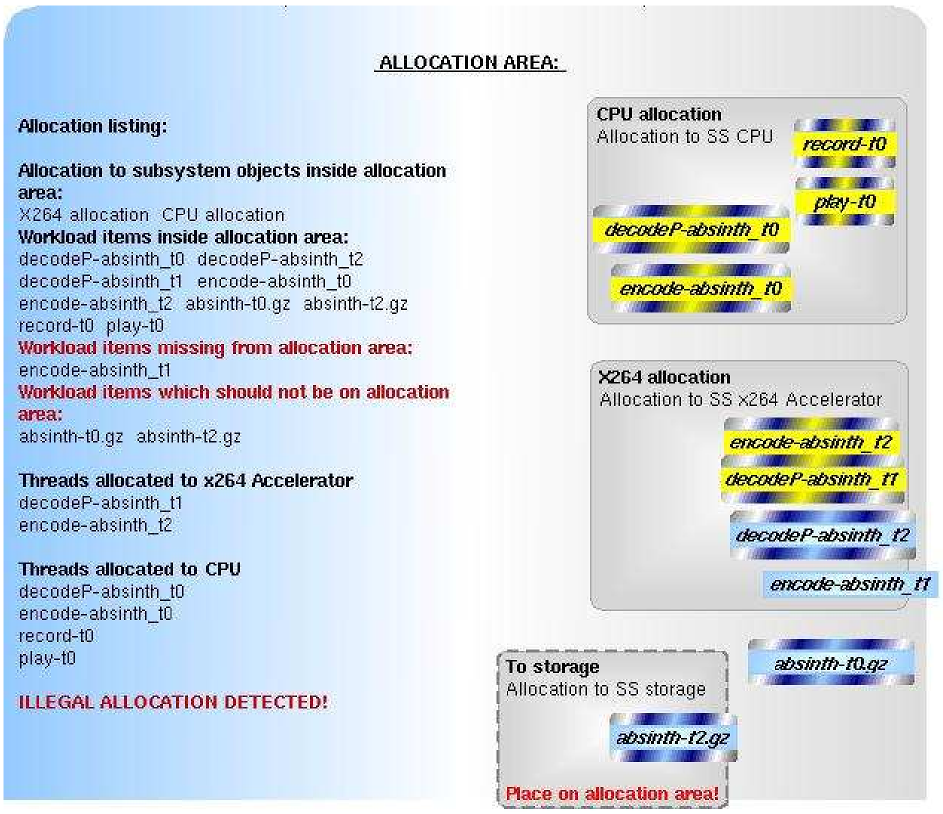
Object locations on a diagram can be utilized with the used DSM tool for building the allocation DSL. The links between application and platform elements can be based on object locations. The allocation DSL uses the *Allocation Area* and the *Allocation to Resource* allocation objects. The application elements are items of the workload, and the *Workload Thread* objects and the *Workload Thread Group* objects are used in the allocation diagram in addition to the allocation objects. The location of workload objects with respect to the allocation objects defines the allocation.

There are several ways to bring workload item objects to the diagram. They can be picked from the object list, which shows existing objects, or they can be created manually from the scratch. They can also be generated according to corresponding workload traces or according prompt input.

The MERL generator producing the allocation file detects all the *Allocation to Resource* objects that are placed on the *Allocation Area* object. The generator also detects items of the workload on top of the *Allocation to Resource* objects and forms the allocation file accordingly. When a workload item is on the *Allocation to Resource* object, which has the valid resource property and the *Allocation to Resource* object is on the *Allocation Area* object, the workload item is correctly placed. Warnings are generated on the allocation objects when the objects are placed wrongly. The *Allocation Area* object contains also a listing of the allocation, which changes according to the locations of the other objects in the diagram. The generator producing the listing can also print warnings, because the *Allocation Area* object has information of the workload items and the platform resources, which can be used in composing the allocation.

Dynamic symbols are used as guidance for the designer towards valid allocations. The symbols of workload objects are made dynamic and the symbols are changing during the allocation work depending how they are placed. A correctly placed workload item has symbol, which is emphasized with yellow. Wrongly placed workload item has two different symbols.

Fig. 4 shows an example allocation, modeled in a diagram presentation of the *Allocation Modeling Graph*. It contains the *Allocation Area object*, *Allocation to Resource* objects and *Workload Thread objects*. The allocation in it contains errors, which are reported in the allocation listing.


ALLOCATION AREA:

Allocation listing:

Allocation to subsystem objects inside allocation area:
X264 allocation CPU allocation
Workload items inside allocation area:
decodeP-absinth_t0 decodeP-absinth_t2
decodeP-absinth_t1 encode-absinth_t0
encode-absinth_t2 absinth-t0.gz absinth-t2.gz
record-t0 play-t0
Workload items missing from allocation area:
encode-absinth_t1
Workload items which should not be on allocation area:
absinth-t0.gz absinth-t2.gz

Threads allocated to x264 Accelerator
decodeP-absinth_t1
encode-absinth_t2

Threads allocated to CPU
decodeP-absinth_t0
encode-absinth_t0
record-t0
play-t0

ILLEGAL ALLOCATION DETECTED!

Fig. 4. Allocation is illegal because two workload items, irrelevant to selected workload, are placed on the allocation area and one relevant item of workload is not inside allocation objects.

5 Modeling with ABSOLUT DSL

The updated workflow of the ABSOLUT modeling enhanced by the developed DSLs is described using a video player as a demonstration case.

5.1 ABSOLUT workflow with DSL

The ABSOLUT workflow presented earlier in Fig. 1 does not change dramatically because of DSLs. The Y-chart flow is still the basis of the modeling method. The main changes are related to the modeling phases, which are enhanced with the DSLs. The developed front-end can also be used for performance simulations and simulation result analysis but this does not change the overall workflow.

The workload modeling DSL gathers all workload modeling information to one diagram. The workload modeling can be therefore managed more efficiently. Different versions of workload models and workload traces can be generated in a controlled way and stored in an appropriate location.

The platform modeling DSL speeds up the composition of platform models from the ABSOLUT model library components and makes it less error prone by avoiding manual editing of XML documents. The platform description generator produces an XML file, which ABSOLUT tools need to produce the platform source code.

Allocation DSL enables drag and drop like method for allocation of workload items to platform resources. Workload items can be imported which reduces effort. It guides the designer and alerts from illegal allocations. In addition to the graphical allocation, a textual allocation listing is visible during the allocation phase. Allocation description files are generated in a format suitable for the ABSOLUT tools.

The generators are used to start ABSOLUT tools and compilers, which produce the simulation model. There is also a generator, which runs the performance simulation.

5.2 Case study

The video player case study has been made with the described front-end. The player uses H264 high definition video coding which is used in high quality mobile devices. The ABSOLUT workload model was generated from the open source FFMPEG [17] source code. The workload generation DSL was used to set the *Workload Model* configuration and single and two thread configurations of the *Workload Trace* object. Workload traces were generated by the generator that is used to execute the workload models.

Our test case used an OMAP4 like platform [18]. The platform model was composed from platform modeling elements with our platform modeling DSL. The platform model is a simplified version of the OMAP4 but e.g. includes the dual-core CPU for testing of different allocations. The XML description of the modeled platform was produced with the platform description generator.

Two different allocations were made with the allocation DSL so that the effect of changing the workload allocation could be detected. The workload items were imported and allocation objects instantiated. Then the allocation was composed and the allocation description files generated. The same was done for both of the workloads.

SystemC simulations were performed with the OMAP4 platform model and two pairs of allocation files and workload models. The utilization of the ARM cores was the property, which was measured from the simulations. The single-threaded version of the application utilizes the Core 0 100 %, but the Core 1 is not used at all. In the dual-threaded version, the load is evenly divided to both cores, which shows that platform capacity is nicely harnessed. The observed utilization rates for the dual-core CPU of the multi-threaded case are presented in Figure 5.

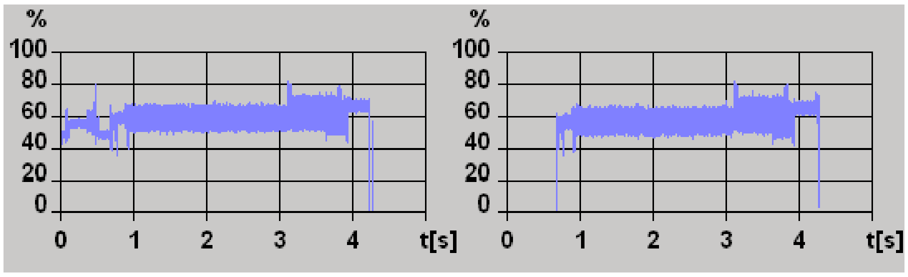


Fig. 5. Data processing load of both cores in the two thread video coding case.

6 Conclusions

This paper describes a way of utilizing the DSM method on the domain of virtual system modeling. The resulting prototype DSL set can be used as a modeling and simulation front-end to the ABSOLUT virtual system modeling tools.

Our experience of DSM was near to zero when the work began and notion of DSM has become clearer in the course of the work. The experiences of the DSM and from the used DSM tool are positive. The DSLs were developed incrementally in smalls steps and they were tested with example data. All three DSLs have evolved steadily without major tool or DSM approach related problems.

The developed DSLs improve the usability of the ABSOLUT, which so far has been used without a graphical front-end. Especially the learning curve shortens due more user-friendly modeling. In particular, the DSL front-end makes modeling easier for designers who are not experienced with SystemC [10] and TLM [11].

The phases of the ABSOLUT virtual system modeling - workload modeling, platform modeling and allocation modeling - were carried out with the developed DSL in a video player case study. The performance simulation was carried out for two system models from which the performance data was recorded. According to our experiences, the DSL-aided workload, platform and allocation modeling is a workable idea.

Our work continues with the refinement of the ABSOLUT DSL. The usage of a DSM tool for simulation observation and simulation result analysis front-end is also an interesting research direction that has already been pretested. There are also possibilities to explore how this sort of DSM approach suits to other embedded system modeling phases.

Acknowledgements. This work is supported by the European Commission and Tekes – the Finnish Funding Agency for Technology and Innovation - under the grant agreement ARTEMIS-2010-1-269362 PRESTO.

The adoption of DSM principles and environment was guided by experts of MetaCase. Especially Mr. Janne Luoma contributed in solving issues faced during the work. Dr. Juha-Pekka Tolvanen receives also our gratitude for professional comments.

7 References

1. Gerstlauer, A., Chakravarty, S., Kathuria, M., Razaghi, P.: Abstract System-Level Models for Early Performance and Power Exploration. In 17th Asia and South Pacific Design Automation Conference, pp. 213-218. IEEE (2012).
2. Kreku J., Hoppari M., Kestilä T., Qu Y., Soininen J.-P., Andersson P., Tiensyrjä K. Combining UML2 Application and SystemC Platform Modelling for Performance Evaluation of Real-Time Embedded Systems, 18p. EURASIP Journal on Embedded Systems. Hindawi Publishing Corporation (2008).
3. Sprinkle, J., Mernik, M., Tolvanen, J-P., Spinellis, D., What Kinds of Nails Need a Domain-Specific Hammer?, IEEE Software, July/Aug, 2009.
4. DSM Forum, http://www.dsmforum.org/ (2012).
5. Kelly, S., Tolvanen, J.-P.: Domain-Specific Modeling: Enabling full code generation, Wiley-IEEE Computer Society Press (2008).
6. Domain-Specific Modeling with MetaEdit+, http://www.metacase.com (2012).
7. Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling, 685 p.. John Wiley & Sons, Inc. (1991).
8. European EDA Roadmap. Technical report, 352 p. CATRENE (2009).
9. Kreku, J., Tiensyrjä, K., Vanmeerbeek, G.: Automatic workload generation for system-level exploration based on a modified GCC compiler. In Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 369-374. IEEE (2010).
10. Grötker, T. and Liao, S. and Martin, G. and Swan, S.: System design with SystemC. Springer, 2002.
11. SystemC Transaction-level Modeling Standard, TLM-2.0, http://www.accelera.org (2012).
12. DSM Tools, http://www.dsmforum.org/tools.html (2012)
13. Eclipse Modeling Project, http://www.eclipse.org/modeling/ (2012).
14. CoFluent Studio, http://www.cofluentdesign.com (2012).
15. Wind River Simics, http://www.windriver.com/products/simics/ (2012).
16. Tolvanen, J.-P.: Domain-Specific Modeling: How to Start Defining Your Own Language, http://www.devx.com/enterprise/Article/30550 (2006).
17. FFmpeg, http://www.ffmpeg.org/ (2012).
18. OMAP Mobile Processors, http://www.ti.com/ (2012).