Model Driven Engineering for MPSoC Design Space Exploration

Marcio F. da S. Oliveira, Eduardo W. Brião, Francisco A. Nascimento, Flávio R. Wagner

Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
e-mail: mfsoliveira@inf.ufrgs.br

ABSTRACT

This paper presents a Model Driven Engineering approach for MPSoC Design Space Exploration (DSE) to deal with the ever-growing challenge of designing complex embedded systems. This approach allows the designer to automatically select the most adequate modeling solution for application, platform, and mapping between application and platform, in an integrated and simultaneous way and at a very early design stage, before system synthesis and code generation have been performed. The exploration is based on high-level estimates of physical characteristics of each candidate solution. In an experimental setting, the DSE tool automatically performs four design activities: it selects the number of processors, maps tasks to processors, allocates processors to bus segments, and sets the voltage of each processor. Experimental results, extracted from a DSE scenario for a real application, show that the proposed estimation and exploration approach may find a suitable solution regarding the design requirements and constraints in a very short time, with an acceptable accuracy, without relying on costly synthesis-and-simulation cycles.

Index Terms: Design space exploration, multi-processor system-on-chip, model driven engineering.

1. INTRODUCTION

With the rising complexity of embedded software and the power/thermal constraints, MPSoC (Multi-Processor System-on-Chip) platforms have emerged as the only feasible solution to meet the strong constraints imposed to embedded system design, such as pressures to reduce the time-to-market of new products as well as system energy consumption. Dealing with this ever-growing challenge only by designer's expertise is not feasible. Observing a design example where 17 tasks must be allocated to a six-processor platform with four different voltage settings for each processor, resulting in a design space with more than 100,000 alternatives, is enough to realize that Computer-Aided Design (CAD) tools are imperative to automate one of the most costly design activities, namely the Design Space Exploration (DSE) of alternative solutions.

Two approaches have been proposed to cope with this problem. In first place, meet-in-the-middle strategies, such as Platform-based Design (PBD) [19], are used to maximize the reuse of pre-designed components and to achieve the best customization of the design according to system requirements. This influences even more the need for automated DSE tools. Secondly, the new generation of embedded system design tools relies on the Model Driven Engineering (MDE) approach [21] to raise the design abstraction level and to provide mechanisms to improve the portability, interoperability, maintainability, and reusability of models. In addition, MDE helps to abstract platform complexity and to represent different concerns of the system [10].

Moving the development focus from implementation to model suggests the support to a fast DSE in the early design steps, where the design effort is now concentrated and modeling decisions can lead to substantially superior improvements. In this context, this work proposes an MDE approach using a UML-based estimation tool called SPEU (System Properties Estimation with UML) [16] and an automatic multi-objective DSE mechanism implemented by the H-Spex (High-Level Design Space Exploration) tool. Our MDE approach is supported by the ModES framework [14], which provides efficient meta-models to capture the system structure and behavior, representing system concerns in separated dimensions - application, platform, mapping, and implementation. Moreover, ModES provides a transformation engine, which interacts with the H-Spex tool to allow both the verification of requirements and the final model generation after the exploration step. These model transformations are actions performed on the models, following a set of rules produced by H-Spex and/or extracted from system requirements specified in the models. Following the proposed approach, a designer can automatically select the most adequate modeling solution for application, platform, and mapping between application and platform, in an integrated and simultaneous way and at a very early design stage, before system synthesis and code generation have been performed. The exploration is based on high-level estimates of physical characteristics of each candidate solution, performed directly from high-level UML models.

In an experimental setting that validates our approach, we consider a platform with up to six processors, each one with four different voltage settings and mapped to two distinct bus segments. In this context, H-Spex automatically performs four design activities: it selects the number of processors, maps tasks to processors, allocates processors to bus segments, and sets the voltage of each processor. Experimental results, extracted from a DSE scenario for the design of a real application, show that the proposed estimation and exploration approach may find a suitable solution regarding the design requirements and constraints in a very short time, with an acceptable accuracy, without relying on costly synthesis-and-simulation cycles. Besides, the use of an MDE approach promotes the reusability of application and architecture models and allows a designer to perform early DSE.

The remaining of this paper is organized as follows. Section 2 discusses related work on model-based approaches. Section 3 introduces the basic concepts of our DSE approach. Section 4 presents the required infrastructure for DSE, including the MDE infrastructure and a model-based estimation tool. The DSE method is presented in Section 5. A real case study, which illustrates and validates our DSE method, is described in Section 6. Finally, Section 7 draws main conclusions and proposes future research directions.

2. RELATED WORK

There are many recent research efforts on embedded systems design using an MDE approach. The adoption of model-based design and the independent specification of platform/application using UML have been vastly investigated.

As a complete environment for DSE, the MILAN [1] / DESERT [15] framework is worth of mention. The focus of MILAN is on the simulation of embedded systems, so that it evaluates pre-selected candidate solutions. The DESERT tool uses models of aggregated system sub-components and constraints to automatically compose the embedded system through Ordered Binary Decision Diagrams and based on a complete pre-characterization of components.

The ARTEMIS [18] / SESAME [17] framework provides methods for modeling and simulation at different abstraction levels, aiming at an automatic DSE of heterogeneous embedded SoCs for multimedia applications. However, this framework does not use MDE, relying instead on lower level languages such as C/C++ and a specific API to represent a Kahn process network. Besides, the designer must specify separated models for implementation and evaluation.

Metropolis [2] is an infrastructure for electronic system design, in which tools are integrated through an API and a common meta-model. Following the platform-based approach and the Y-chart methodology, the Metropolis' infrastructure captures application, architecture and mapping using a proposed UML-platform profile [5]. Furthermore, its infrastructure is general enough to support different models of computation (MoCs) and accommodate new ones. Non-automatic support for design space exploration is provided by Metropolis, which proposes an infrastructure to integrate different tools. Nevertheless, the current simulation and verification tools integrated into Metropolis and the proposed refinement process can be used to perform some manual architectural explorations (task mapping, scheduling, hardware/software partitioning) and component configurations. Moreover, the refinement process allows the explicit exploration of application algorithms, that implement a higher-level specification.

The DaRT (Data Parallelism to Real Time) [4] project proposes an MDA (Model-Driven Architecture) - based approach for SoC design that has many similarities with our approach in terms of the use of meta-modeling concepts. The DaRT project defines MOF-based meta-models to specify application, architecture, and software/hardware associations and uses transformations between models as code transformations to optimize an association model. In doing so, it allows the re-factoring of an application model in order to better match it with a given architecture model. In DaRT, no DSE strategy based on these transformations is implemented, and the focus is mainly the code generation for simulation at TLM (Transaction Level Model) and RT (Register Transfer) levels.

Koski [8] is a UML-based framework to support MPSoC design. It is a library-based method, which implements a platform-based design. Koski provides tools for UML system specification, estimation, verification, and system implementation on FPGA. Following the design flow, the application, architecture and the initial mapping are specified as UML 2.0 models. A UML interface handles these models and generates an internal representation, which is used for architectural exploration. The architectural exploration is performed in two steps; the first one is static, fast and less accurate; the second one is dynamic. At the end of the design flow, the UML models are used to generate code, and the selected components from the platform are linked to build the system.

Compared to our approach, no related work takes advantage of the MDE notion of transformation between models to represent, delimitate, and explore the design space. Furthermore, the proposed DSE methodology is combined with SPEU, a model-based estimation tool, allowing a designer and/or the H-Spex tool to verify possible implementations for a given application on a specified platform at a high abstraction level.

3. EXPLORATION APPROACH

As most DSE approaches, we follow the Y-chart [10] to represent the complete design space that will be explored. Three domains are proposed to represent the design space – Application, Platform, and Mapping – for which a designer can explore a large number of configurations. A fourth domain, the Implementation one, is proposed to represent the decision resulting from the DSE process. Figure 1 illustrates these concepts.

In the Application Domain, the designer expresses the application concepts using a modeling language, such as UML. A designer may investigate issues such as the distribution of responsibilities between classes, the encapsulation of functionality inside objects, the aggregation of objects inside independent threads, and strategies for the interaction between objects.

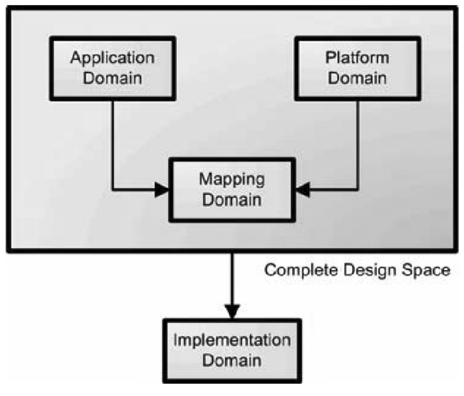

Figure 1. Design space exploration domains.

The platform components are represented in the Platform Domain. In this domain, a designer can explore different implementations of the same component, the allocation of processors, the individual parameter settings of each selected component, the communication structure, and others.

The Mapping Domain expresses the mapping between application and platform concepts. In this domain, a designer may explore the mapping of tasks to processors, the mapping of hardware resources to a communication structure, and the software binding. The result of the DSE process is expressed in concepts at the Implementation Domain, which contains the necessary information to generate/synthesize an embedded system.

4. DESIGN SPACE EXPLORATION INFRASTRUCTURE

The pre-condition to implement the automatic support to design space exploration is the utilization of appropriate models to represent the problem and the solution for each activity. The infrastructure, which allows the interaction between the automatic exploration tool and the user, is also important. This infrastructure allows the specification of models, exploration parameters, and system constraints / requirements (functional and non-functional requirements). It also supports the integration between exploration activities, including models and sharing of results.

An important requirement in this infrastructure is the automatic mechanism for the evaluation of alternative solutions, which must be performed at a high-level of abstraction in order to be effective. If assessing each candidate solution would require its detailed synthesis and cycle-accurate simulation, design time would be prohibitive.

Therefore, the proposed work is integrated in an efficient infrastructure, which provides the required support for MDE-based design of embedded systems. Moreover, a model-based estimation tool is integrated to this infrastructure in order to allow early and fast evaluation of design alternatives, still at a very high abstraction level.

A. MDE Infrastructure

Based on the presented DSE concepts, the ModES (Model-driven Embedded System design) framework [14] has been implemented to provide an MDE infrastructure. This infrastructure is composed by a set of four meta-models, one for each domain: Internal Application, Internal Platform, Mapping, and Implementation Meta-model. All concepts in the meta-models are specified as UML classes and then converted into Eclipse/EMF models [6].

The MDE infrastructure provides a Java API, automatically generated by the EMF tool, to handle meta-models instances. The models specified by the designer are loaded into the MDE infrastructure by means of model-based transformations implemented as QVT (Query, Views, Transformations) resources, standardized by OMG [13].

The designer must specify the application models using any UML 2.0 compliant tool that allows the generation of the appropriate XMI files. The most important UML diagrams used in our approach to specify the system are Use Cases and Sequence, Class, and Deployment Diagrams. Using the UML profile for Scheduling, Performance and Time (UML-SPT) [22], the designer can extend the application model with constraints or requirements. The structural and behavioral aspects of the Application Model are captured into the Internal Application Meta-model (IAMM) (shown in Figure 2).

Using the Internal Application Meta-model, a system specification captures the functionality of an application in terms of a set of modules that are composed of concurrent communicating processes and/or sub-modules. The module behavior is captured in terms of actions represented by a Control Data Flow Graph (CDFG) that captures the data and control flow between the actions. The Actions were removed from Figure 2, due to the reduced space. The CDFG corresponds to the UML actions of the scenarios (sequence diagrams) that are related to the process, according to its active object. The adopted concepts correspond to CSP-like languages, which are able to express any kind of concurrent, distributed system.

In a platform-based design environment, a large number of hardware and software components are provided and can be reused in the system development. To evaluate different solutions at a high abstraction level and perform DSE, the reused components must be pre-characterized in terms of performance, energy, memory footprint, and others.

The software component characterization is performed after the component code is compiled for the target architecture, since at this time a simulation/estimation tool can capture architectural information with high accuracy. Likewise, hardware components are characterized for a specific technology.

In the ModES framework, the information about the characterization is stored in an instance of the Internal Platform Meta-model, whose data model is based on the General Resource Model from UML-SPT, identifying the services offered by the platform and the related quality-of-service values. As for application models, in order to have a standard representation, the Platform Meta-Model is translated into an Internal Platform Meta-Model (IPMM), shown in Figure 3.

Providing an extensive component repository requires a significant effort. However, normally a large amount of system components can be reused from different component providers or as a sub-product from previous system developments [20].

Therefore, the platform repository creation is based on the accumulation of components produced or acquired in previous product developments. Currently, the platform repository contains information on processing units (different versions of a Java microcontroller [7]), scheduling and timer services, a real-time communication API [23] implemented on top of the Java microcontroller, and a math and image-processing library.

As additional architectural aspects, information is stored about the supported data types and instruction set, such as size, number of execution cycles, and energy consumption for each instruction. Aspects of program and data memory allocation are also considered, including dynamic frame allocation, reserved memory, data type, and method allocation rules.

To add new IP (intellectual property) resources to the platform repository, the IP provider must attach this architectural information to his/her IPs. After the repository is populated with adequate information on the platform components, the designer just needs to map the application into the platform model. This allows the evaluation of a general application model (Platform Independent Model – PIM), when

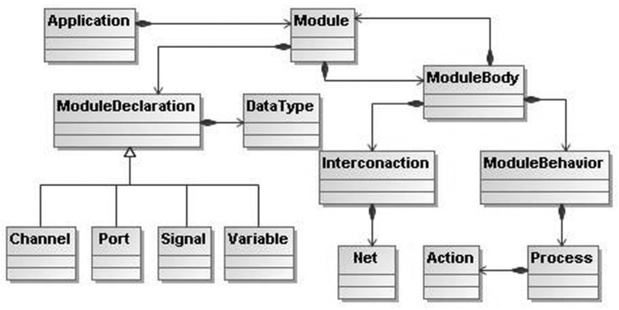

Figure 2. Internal Application Meta-model (IAMM).

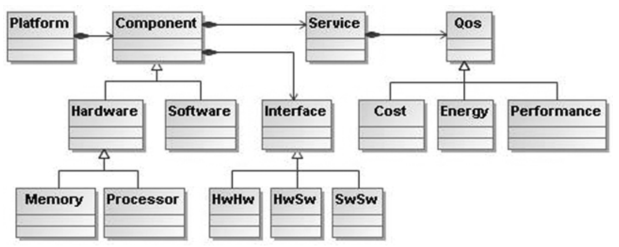

Figure 3. Internal Platform Meta-model (IPMM).

mapped to a specific platform that corresponds to the Platform Specific Model (PSM), according to the MDA [11] approach.

The Mapping Meta-model describes the rules used to transform instances of the Internal Application Meta-model and Internal Platform Meta-Model into an instance of the Implementation Meta-model, as shown in Figure 4.

In the proposed approach, the Mapping Meta-model is based on a set of transformations, where each transformation is composed by a set of rules that limit the possible mappings between application and platform, thus guiding the H-Spex tool or the designer to build a candidate Implementation Model. This allows the evaluation of possible implementations during the DSE phase. Besides, the task mapping, processor allocation, and software binding can also be expressed as UML deployment diagrams, reflecting a design restriction on the DSE process.

A transformation engine can be invoked to traverse the set of transformation rules, in order to select and apply the ones that are enabled at that moment and to produce a hardware/software mapped architecture in the form of an Implementation Model. Our transformation engine is implemented using MDDi-QVT (Model Driven Development integration - Query, View, Transformation) [12], which is an open-source implementation of the QVT standard. MDDi-QVT provides a QVT parser, which reads a textual QVT specification and builds a model conforming to the QVT meta-model [13], and a QVT compiler that generates an API in Java from a QVT model, which can be used to perform transformations between models. Thus, in order to use the current version of MDDi-QVT, we generate a textual QVT specification file from the application, platform, and mapping models conforming to our meta-models. Then we apply the QVT parser and compiler to obtain a Java API. The H-Spex tool calls this API, which implements the transformation engine.

The Implementation Meta-model allows the representation of the models that can implement the system specification without violating the system requirements. Figure 5 shows the Implementation Meta-model class diagram.

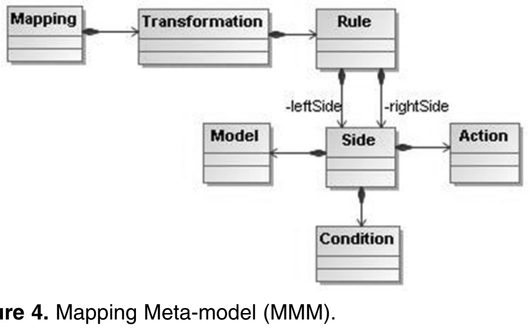

Figure 4. Mapping Meta-model (MMM).

The Implementation Meta-model can be described as a list of selected platform components, application components, and associations between them. Furthermore, it also represents the design decisions about component configuration, task mapping, and resource allocation. An instance of the Implementation Meta-model is used, together with instances of other proposed meta-models, by a tool that generates partial embedded software source code, scripts for a hardware synthesis, and system deployment scripts. As explained in the next section, other tools, such as an estimation one, can use an instance of the Implementation Meta-model in order to estimate the properties of a candidate implementation model before the development process is continued.

B. Model-based Estimation Tool

The SPEU tool [16] provides analytical estimates about physical system properties (execution cycles, energy/power consumption, volume of communication data, and memory footprint). These properties are directly obtained from instances of the meta-models, with estimation errors as low as 5%, when the reuse of repository components is largely employed by a PBD approach.

The estimation is performed by using the information extracted from UML application structure/behavior models and stored in the Internal Application Meta-model as a CDFG (Control and Data Flow Graph). To improve the estimation accuracy, the information specified in the Platform Model is used to compute the costs of pre-designed components and added in the final estimation. An ILP (Integer Linear Programming) formulation is used for the identification of best-case and worst-case execution paths on the CDFG. In order to reduce the complexity, each ILP formulation is solved in a hierarchical fashion, such that each task has one ILP formulation to be solved. This avoids problems for MPSoCs with a large number of tasks. This estimation tool allows H-Spex to rapidly evaluate each candidate solution during the DSE process, without depending on costly synthesis-and-simulation evaluation cycles. After that, the estimated properties are used to compute the solution cost in a parameterized multi-objective function, which is optimized by the exploration algorithm.

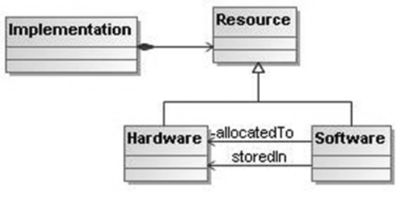

Figure 5. Implementation Meta-model (IMM).

5. EXPLORATION METHOD

The model-based exploration method is implemented by H-Spex (High-level design SPace EXploration), a tool for automatic multi-objective design space exploration. H-Spex is supported by MoDES, which provides the required MDE infrastructure. H-Spex is also supported by SPEU (System Properties Estimation with UML), which provides the estimation values needed for evaluation of alternative solutions during design space exploration.

The DSE mechanism of H-Spex adopts a heuristic approach to investigate the space of possible architectural solutions, which is currently based on simulated annealing [9]. The implemented simulated annealing uses a probabilistic perturbation function, whose values were experimentally selected. The perturbation function is responsible for looking for a solution that minimizes the communication, energy, power, performance, memory footprint, or any combination of these properties, under hard real-time constraints, meeting design restrictions such as the processing capacity of each processor (which depends on its frequency/voltage setting). These metrics are used to calculate the solution cost using an objective function, which the simulated annealing algorithm tries to minimize. H-Spex reads the constraints specified using UML-SPT in the Application Model (and captured by the Internal Application Model) and compares them with the results obtained by SPEU, the model-based estimation tool. At each moment, H-Spex takes design decisions and incorporates the corresponding information in the conditions of the transformation rules of the mapping model. Then, H-Spex invokes the transformation engine provided by the MDE infrastructure, so that it produces candidate Implementation Models, which will be evaluated in order to select the final solution. Figure 6 shows the interaction between H-Spex and the infrastructure for design space exploration.

The current DSE prototype selects the most appropriate solution by taking several design decisions: it selects the number of processors in the MPSoC, partitions tasks between processors, maps processors to a communication structure (currently, only a segmented bus), and schedules the voltage level for each processor. Therefore, H-Spex is automatically exploring both the Platform and Mapping Domains. The Application Domain could also be explored, but this is currently manually performed by the designer, as described in [16].

The exploration flow is depicted in Figure 7. Firstly, H-Spex randomly generates an initial solution. When all design decisions for a candidate solution were taken, the transformation engine is invoked to traverse the set of transformation rules of the Mapping Model, so as to select and apply the ones that are enabled at that moment, in order to produce a platform-specific model in the form of an Implementation Model. This model is the input to SPEU, which supports model evaluation for early and fast DSE.

6. CASE STUDY

A DSE scenario for the design of a real application, concerning the automated control of a wheelchair with functions such as movement control, collision avoidance (ultrasound and stereo vision), and navigation, has been developed. The application model contains 17 tasks, which have communication dependences between them. The platform model is based on a Java microcontroller [7], upon which an API and operating system components [23] for real-time behavior and communication support are available. Figure 8 illustrates the task graph extracted from the Application Model, which has been specified in UML and loaded into the Internal Application Metamodel.

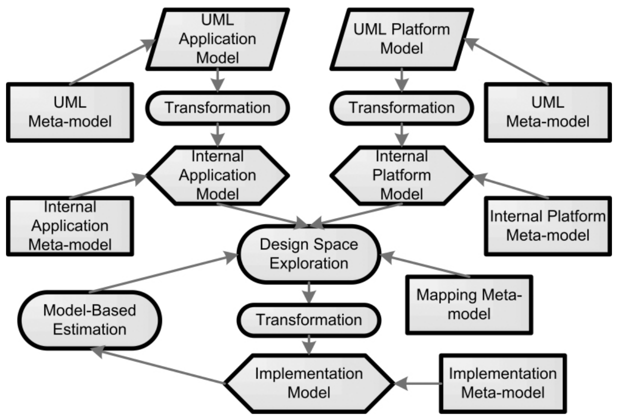

Figure 6. Tools integration for MDE design space exploration.

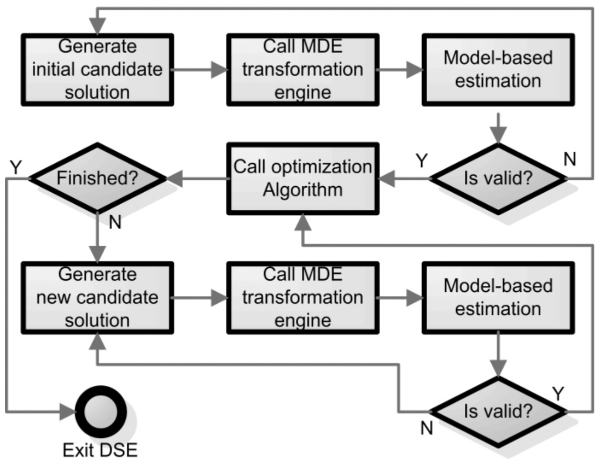
Generate initial candidate solution
Call MDE transformation engine
Model-based estimation
Y
Finished?
Call optimization Algorithm
Y
Is valid?
N
N
Generate new candidate solution
Call MDE transformation engine
Model-based estimation
Y
Is valid?
N
Exit DSE

Figure 7. Exploration algorithm.

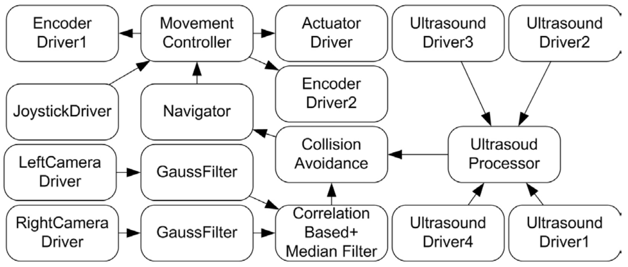

Figure 8. Application model task graph.

To implement the movement control, several components provided by the platform can be reused, such as a mathematical library to solve the control equations, a real-time Java API, and RTOS components [23]. Consequently, the platform aspects considered in the estimation methodology also take into account costs that are related to the real-time scheduler. Table 1 shows some of the available components of the platform model and corresponding services.

The component *Math:API* has many services, as for example *sin:Service*, *cos:Service*, *atan:Service*, and *sqrt:Service*. All these services are used for the implementation solutions. The *RestoreContext:Service* is a service of the scheduler *EDFScheduler:OS*, which restores the processor state and gives control back to a previous thread. The *waitForNextPeriod:Service* passes the control from a real time thread to the OS at the end of its execution. The *FemtoJavaMCRTC* (Multi-cycle, RealTime Clock) is one of the versions of a Java microcontroller available in the platform [23]. It has instructions that are not available in the simplest version of the Java microcontroller, but that are needed to support the *RealtimeThread:API*.

Table 2 lists some of the transformation rules of the mapping model. The transformation rules specify the possible mappings from elements of the wheelchair application model to elements of the FemtoJava platform, including the actions that will generate elements in the implementation model.

As specified by the transformation rules, the modules can be implemented either by a simple FemtoJava or by a multi-cycle real-time FemtoJava. In the implementation model, a memory component will be instantiated for program code and data, and a bus will be instantiated to connect the processor to the memory. Other transformation rules state that: a) processes in the application can make use of the *Math* and *RTThread* APIs; b) actions in the thread of the processes may use services to suspend the execution (*waitForNextPeriod* and *waitForCycles* services); c) actions can have access to static and dynamic object attributes (*get/set Static/Dynamic ObjectField* services); d) mathematical operations can be implemented by the *Math API*; and e) scheduling of processes and threads can be implemented by different kinds of schedulers from a given operating system.

Given the platform, application, and mapping models, the DSE process takes design decisions about the partitioning, allocation, and binding of application elements to platform elements and incorporates the information on the mapping model.

H-Spex selects the number of processors and maps tasks to them in order to minimize the overall

Table 1. Part of Platform Model

Component	***Service***	***Prog. mem. (bytes)***	***Data mem. (bytes)***	***Perf. (cycles)***	***Energy (switching activity)***
Math:API	Sin:Service	14	4	53	73,815
EDFScheduler: OS	RestoreContext: Service	4	0	883	8,117.3
Realtime Thread: API	waitNextPeriod: Service	243	13	5721	1,254.3
Femtojava MCRTC: Processor	int_tf0:Service	2	32	124	187.45

Table 2. Part of Mapping Model

Source IAMM	***Target IPMM***	***Condition***	***Action***
mod: Module	p:Processor, m:Memory, b:Bus	p.name=FemtoJava32 or p.name=FemtoJavaMCRTC	new Processor( ), new Memory( ), new Bus( )
proc: Process	a:API	a.name=Math or a.name=RTTrhead	new API( )
threadctrl: Action	s:Service	s.name=waitNextPeriod or s.name=waitForCycles	new Service( )
interaction: Action	s:Service	s.name=interactionStatic or s.name= interactionDynamic	new Service( )
getObjectField: Action	s:Service	s.name=getStaticObjectField or s.name=setStaticObjectField or s.name=setDynamicObjectField	new Service( )
mathoper: Action	s:Service	s.name=sqrt or s.name=sin or s.name=cos or s.name=s.atan	new Service( )
sched: OS	s:OS	s.schedtype=EDFScheduler or s.schedtype=FixedPriority or s.schedtype=RMScheduler	new OS( )

communication costs between tasks. Tasks with a higher communication bandwidth between them tend thus to be grouped in the same processor, if possible. The candidate task partitioning can neither violate the task deadlines nor exceed the processing capacity of each node. If necessary, H-Spex adds another processor to the system. Simultaneously, H-Spex maps processors to the communication structure, such that the overall communication cost is again minimized. Therefore, processors with a higher communication between them (because of the tasks assigned to each one) tend to be mapped to the same bus segment. In order to reduce the energy consumption, H-Spex also selects the minimal voltage for each processor, but avoiding task deadline violations. Realistic energy, voltage, performance, and memory costs for HW and SW platform components and services (such as task scheduling), extracted by using the SPEU model-based estimation tool, have been used to guide the exploration.

The number of design alternatives in this DSE scenario is huge, so that the utilization of an automated DSE approach at very high abstraction level is fully justified.

Figure 9 shows a chart with the final solution found by H-Spex for different combinations of optimization objectives. In this chart, the values of solution properties were normalized with regard to the worst solution, so that they could be presented in the same chart. The “x” axis is organized by objective of optimization - energy, power, memory, energy-power-cycles-memory (epcm), energy-power-cycles (epc), and energy-power-memory (epm).

The DSE results for energy optimization present a good load distribution between the six processors. This parallel execution reduces the number of execution cycles, directly contributing to energy savings. Because of data and code replication in the various processors, the values for memory and power were not optimized. In order to minimize the power, H-Spex mapped simple tasks to the same processor with low voltage settings. Processors with high utilization require higher voltage settings.

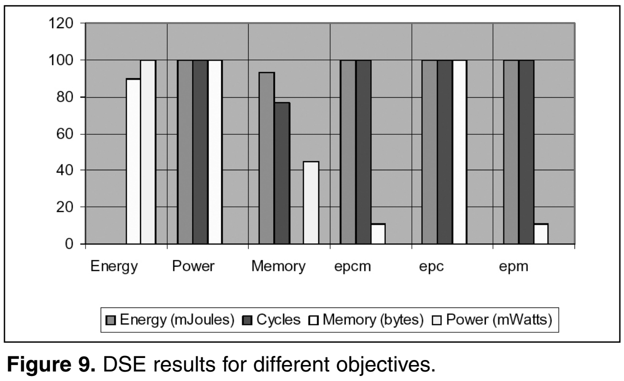

Figure 9. DSE results for different objectives.

In order to reduce the memory size, H-Spex mapped tasks that exchange large amounts of data to the same processor, reducing in this way the data replication and communication structures (packages and messages).

In the last three results in Figure 9, the chart presents DSE results using multi-objective functions. Due to the hard real-time constraints, H-Spex could not reduce the cycles by mapping many tasks to the same processor and thus reducing the communication cost, because this would cause task deadline violations. However, by distributing tasks in different processors and setting low voltages for each processor, the DSE tool could get better results for power, which dominates the optimization process.

Better optimizations could be achieved if the Application Domain was also explored by H-Spex. This exploration could result in reduced values for execution cycles, for instance by changing the interaction between objects and the interaction of the objects with the platform services (reducing service calls), or by looking for a new distribution of responsibilities, in order to arrange the functionalities in a way to improve the parallelism.

Figures 10-13 show the property values estimated for 19 candidate solutions found by H-Spex, by optimizing the objectives energy, cycles, power, and memory, respectively. In each chart, the Y-axis shows the property values for the entire system. The X-axis presents the candidate solutions. The presented results were obtained after evaluation of 1000 candidate solutions. In our experiments, H-Spex takes 1 hour to evaluate each 1000 solutions, using an AMD Athlon 1.8 GHz with 512 Mbytes of RAM and SPEU as the estimation tool.

In order to illustrate the accuracy of the model-based estimation tool, components responsible for the wheelchair movement control were implemented on the target platform using two different solutions. These solutions differ in the objects' structure (number of classes, objects, and threads) and behavior (interaction between application calls to platform services). These changes were selected to emphasize the impact of model decisions on the final system physical properties. The values estimated by the model-based estimation tool were compared to the results obtained through cycle-accurate simulation using the CACO-PS power simulator [3].

Table 3 shows the exact differences between property values of two alternative solutions, obtained by CACO-PS in a cycle-accurate simulation, and the estimated differences, obtained by SPEU. Both performance and energy were estimated in terms of best-case (BC) and worst-case (WC) executions.

The table shows that the estimated differences between property values are very close to the exact dif-

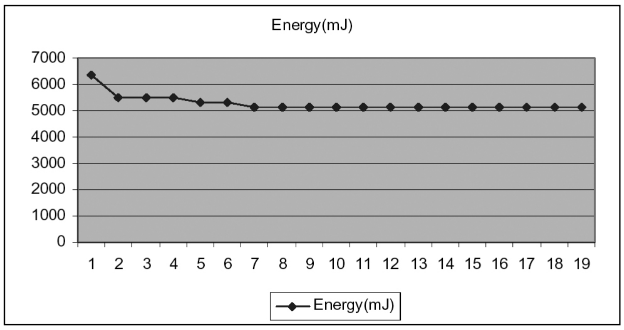

Figure 10. Energy values for 19 candidate solutions.

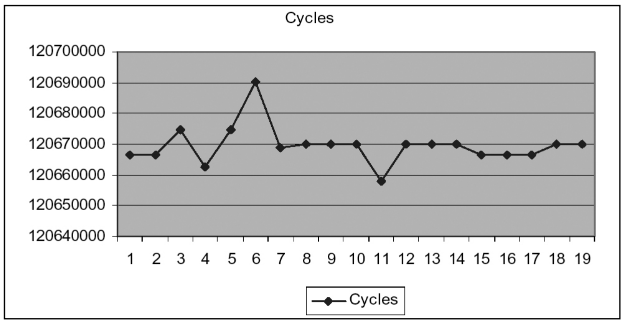

Figure 11. Execution cycles for 19 candidate solutions.

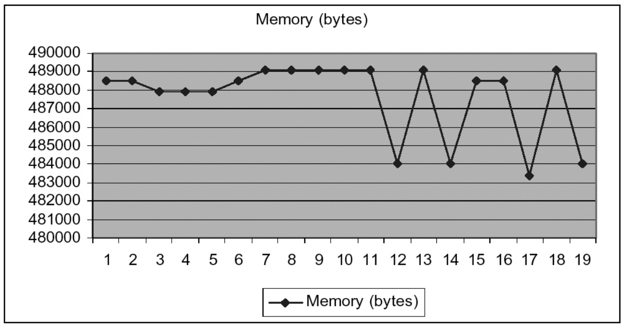

Figure 12. Memory size for 19 candidate solutions.

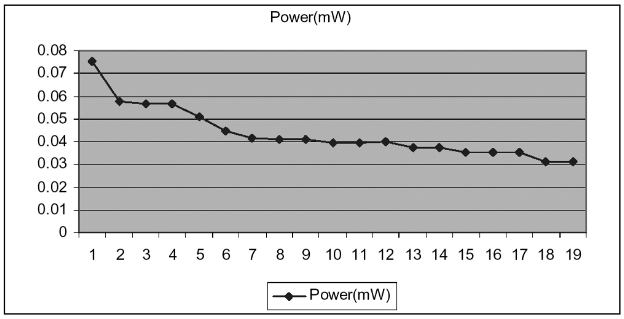

Figure 13. Power values for 19 candidate solutions.

ferences. The time spent to simulate each solution at cycle-accurate level was approximately 23 minutes, while the model-based estimation provided by SPEU spent less then 4 seconds, both times measured with the same host machine configuration described before.

The results in Table 3 show that the second alternative solution is better than then first one in all system properties according to both the estimated and exact evaluation methods. Observing the results, we can conclude that the model-based estimation tool provides quantitative information about physical system properties to support a very fast and reasonably accurate exploration of the impact of these choices on the final system implementation.

Table 3: Differences between two alternative solutions

	Second x First Solution	
Property	**Estimated**	**Exact**
Program memory	-69.83%	-66.98%
BC Data memory	-36.47%	-36.08%
WC Data memory	-36.47%	-36.08%
BC Performance	-95.42%	-93.36%
WC Performance	-65.82%	-65.32%
BC Energy	-99.61%	-93.31%
WC Energy	-66.57%	-65.51%

This case study illustrates a design space exploration scenario for a real application presenting a large design space. Through the Model Driven Engineering approach for Multi-objective Design Space Exploration presented in this work, a designer can specify the system using UML models at high abstraction level (without source coding/generation) and look for the best alternative to be implemented, early in the development process, without relying on costly synthesis-and-simulation evaluation cycles.

7. CONCLUSIONS AND FUTURE WORK

A Model-Driven Engineering (MDE) approach for MPSoC design space exploration and the H-Spex tool, which implements this approach, were presented. This approach adopts a meta-modeling infrastructure with specific meta-models for the application, platform, mapping, and implementation domains. The MDE fundamental notion of transformation between models is used to represent and delimitate the design space of possible implementations of the application on the specified platform. A set of transformation rules defines the possible mappings from application into platform. By evaluating these rules, an exploration tool builds a model solution, selecting the number of processors, mapping application tasks to the selected processors, mapping processors to a communicating structure, and reducing the energy consumption by voltage scaling.

To support this approach, the SPEU tool implements a UML-based estimation, which allows the evaluation of models while exploring the design space, in order to find a model that better fulfills the application requirements. This high-level estimation approach allows a fast comparison of the various models without having to generate executable code, hence without any costly hardware and software synthesis/simulation steps.

Experimental results show that the proposed estimation and exploration approach may find a suitable solution regarding the design requirements and constraints in a very short time, with an acceptable accuracy, without relying on costly synthesis-and-simulation cycles. Besides, the use of UML and MDE promotes the reusability of application and platform high-level models.

One of the future directions to be considered is the meta-modeling improvement, in order to represent more complex applications, platforms, and mappings. We also intend to test different optimization algorithms in order to improve the design space search. Moreover, we shall extend the exploration tool, by adding new architectural exploration activities and automating the exploration activities in the Application Domain, as proposed in [16].

REFERENCES

[1] Agrawal, A. et al. “MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems”. LCTES, 2001.
[2] Balarin, F.; Watanabe, Y.; Hsieh, H.; Lavagno, L.; Passerone, C.; Sangiovanni-Vincentelli, A. “Metropolis: An Integrated Electronic System Design Environment”. Computer, Vol. 36, No. 4, 2003.
[3] Beck, A.C et al. “CACO-PS: A General Purpose Cycle-Accurate Configurable Power Simulator”. SBCCI, 2003.
[4] Bondé, L.; Dumoulin, C.; Dekeyser, J.-L. “Metamodels and MDA Transformations for Embedded Systems”. FDL, 2004.
[5] Chen, R.; Sgroi, M.; Lavagno, L.; Martin, G.; Sangiovanni-Vincentelli, A.; Rabaey. J. “UML and Platform-Based Design”. In: Martin, G. (ed.), UML for Real: Design of Embedded Real-Time Systems. Boston, Kluwer Academic Publishers, 2003.
[6] EMF - Eclipse Modeling Framework http://www.eclipse.org/emf
[7] Ito, S. A. et al. “Making Java Work for Microcontroller Applications”. IEEE Design & Test of Computers, Vol. 18, No. 5, 2001.
[8] Kangas, T et al. “UML-based Multi-Processor SoC Design Framework”. Transactions on Embedded Computing Systems, Vol. 5, No. 2, 2006.
[9] Kirkpatrick, S.; Gelatt, C. D.; Vecchi Jr., M.P. “Optimization by Simulated Annealing”. Science, No. 4598, May, 1983.
[10] Kreutzer, K. et al. “System-Level Design: Orthogonolization of Concerns and Platform-Based Design”. IEEE Trans. on CAD of Integrated Circuits and Systems, Vol.19, No.12, 2000.
[11] MDA Guide Version 1.0.1, OMG, doc. n.: omg/2003-06-01, June 2003
[12] MDDi-QVT: Model Driven Development integration-Query, View, Transformation. <http://www.modelware-ist.org>
[13] MOF QVT - QueryViews/Transformations. OMG doc. ptc/2005-11-01, Nov. 2005.
[14] Nascimento, F.A. et al. “ModES: Embedded Systems Design Methodology and Tools based on MDE”. MOMPES, 2007.
[15] Neema, S. et al. “Constraint-Based Design-Space Exploration and Model Synthesis”. EMSOFT, 2003.
[16] Oliveira, M.F.S. et al. “Early Embedded Software Design Space Exploration Using UML-based Estimations”. RSP, 2006.
[17] Pimentel, A.D.; Erbas, C.; Polstra, S. “A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels”. IEEE Trans. on Computers, Vol.55, No. 2, 2006.
[18] Pimentel, A.D. et al. “Exploring Embedded-Systems Architectures with Artemis”. IEEE Trans. on Computers, Vol. 34, No. 11, 2001.
[19] Sangiovanni-Vicentelli, A.; Martin, G. “Platform-based Design and Software Design Methodology for Embedded Systems”. IEEE Design and Test of Computers, Vol. 18, No. 6, 2001.
[20] Shandle, J.; Martin, G. “Making Embedded Software Reusable for SoCs”. EEDesign, March 2002. <http://www.eedesign.com/story/OEG20020301S0104>.
[21] Schmidt, D. C. “Model-driven Engineering”. IEEE Computer, Vol. 39, No.2, 2006.
[22] UML SPT - UML Profile for Schedulability, Performance, and Time. Object Management Group (OMG) doc. ptc/02-03-02, 2000.
[23] Wehrmeister, M.A.et al. “An Object-Oriented Platform-based Design Process for Embedded Real-Time Systems”. ISORC, 2005.