Embedded SW Design Exploration Using UML-based Estimation Tools

Marcio F. da S. Oliveira¹, Lisane Brisolara¹, Flávio R. Wagner¹, Luigi Carro¹²

¹Computer Science Institute, Federal University of Rio Grande do Sul (UFRGS), Brazil
{mfsoliveira, lisane, flavio, carro}@inf.ufrgs.br

²Electrical Engineering Department, UFRGS, Brazil
carro@eletro.ufrgs.br

Abstract. This paper presents an approach for the estimation of memory, performance, and energy from an initial UML specification. It allows the designer to evaluate and compare different modeling solutions, thus supporting design space exploration at a very high abstraction level. An experiment demonstrates our approach, in which an application is modeled in different ways and the solutions are compared using the high-level estimates. Experimental results are presented and demonstrate the effectiveness of the estimates in an early design space exploration. These results present a very small error when software components are reused and their costs are already known, achieving errors that are smaller than 10%.

1 Introduction

The increasing complexity of embedded systems design, which is derived from the amount of functionality that is required from these systems, together with the shortening of the life cycle of embedded products, results in a design scenario where productivity and quality are simultaneously required in order to deliver competitive products. Selic [1] emphasizes that the use of techniques starting from higher abstraction levels is crucial to the design success.

Platform-based design and IP reuse are techniques that improve the design productivity [2]. In platform-based design, design derivatives are mainly configured by software, and software development is where most of the design time is spent. Moreover, in platform-based design, different granularities can be used to provide a rich and pre-characterized library of software and hardware components to rapidly assemble a system. Furthermore, this pre-characterized library reduces the uncertainty about the system properties.

The UML language has gained in popularity as a tool for specification and design of embedded systems. Efforts that describe the use of UML during the different phases of an embedded system design can be found in [3].

It is widely known that design decisions taken at higher abstraction levels can lead to substantially superior improvements. This context suggests the support to a fast design space exploration in the early design steps. However, software engineers,
when developing an application using UML, do not have a concrete measure of the impact of their modeling decisions on issues such as performance and energy for a specific embedded platform.

This paper proposes an approach that allows a better evaluation of modeling solutions, through the estimation of the final system physical characteristics directly from UML models, thus supporting a very early design space exploration. Our approach provides the estimation of memory, performance, and energy consumption for a system modeling solution.

An experiment demonstrates that different UML modeling solutions for the same application result in different physical costs. When components are reused and the costs of their functions are previously known we achieve estimation errors smaller than 10%. It is shown that, by using our approach, the designer can really explore the system software architecture, already at the UML level.

This paper is organized as follows. Section 2 gives an overview of related work. Section 3 introduces our proposal for embedded software design exploration. Section 4 presents the experiment that illustrates a scenario of design space exploration. Section 5 presents main conclusions and discusses future work.

## 2 Related Work

Bernardi et al. [4] propose the automatic translation of state and sequence diagrams into a generalized stochastic Petri net and a composition of the resulting net models in order to reach an analysis goal. This proposal allows the analysis of the system execution time, but the operation times have to be manually included in the model.

An UML profile and an methodology for performance modeling of UML-based systems are proposed by Theelen et al. [5]. The UML models are translated into an executable model specified by POOSL (Parallel Object-Oriented Specification Language). Some behavioral aspects and time delays can be specified in a probabilistic way, through provided libraries. However, the designer needs to go down the abstraction level from UML to POOSL to estimate system properties.

Petriu and Woodside [6] present an approach that uses the UML-SPT profile, where desired values for performance measures are manually added to the model. A performance model is then derived, from which the performance analyses are computed by simulation or by analytical techniques.

Smith et al. [7] propose a quantitative approach for system analysis, called SPE (Software Performance Engineering). SPE focuses on use cases and on the scenarios that describe them. It allows the identification of the workloads that are most significant to the performance. The sequence diagram representing a key scenario is translated into an execution graph. Software and computer resource requirements are then added to this graph and the model is solved.

All previous works lack an efficient estimation method to obtain information related to performance, such as number of instructions or execution time, and usually this information must be manually specified in the models. Moreover, these approaches do not consider important costs for embedded systems, such as memory and energy consumption. This work proposes a multi-variable methodology, where
energy and memory footprint can also be estimated, not only performance. Moreover, our approach is not based on simulation, thus requiring less time to provide estimates and less detail about the system implementation, when compared to other approaches.

3 Embedded Software Design Exploration Approach

We propose an UML-based estimation approach that allows the estimation of the system properties by mapping the application model into a given pre-characterized platform. It can be used to support design space exploration at a very high abstraction level. In order to support the estimation process, rules for modeling the application and a way to represent information about a given platform have been defined. An estimation tool called SPEU has been developed to implement this approach.

The UML diagrams used in the application model are Use Cases, Interaction Diagrams, and Class Diagrams. These diagrams can be decorated with UML-SPT stereotypes. Structural information can be specified in the class diagram, such as multiplicity of values. The behavioral information is captured from the interaction diagrams, in which we can specify interaction, conditional execution, and dependencies between execution scenarios.

Access methods and delegations of these methods are usually employed in the object-oriented programming approach. In order to identify these methods, some restrictions are imposed to the designer: (i) the “get” or “set” prefix must precede the class field name that has its method invoked; (ii) if the class does not have a field specified in its method name, this means that the method delegates an access to another class; (iii) the suffix “Content” should be used, indicating that the method is accessing the structure like vector or matrix positions. A distinction between these methods is important because they have different costs.

The platform model can contain information on processor architecture, API, operating system, application components, and device drivers. Currently, for the platform model, we use architectural aspects, scheduler and timer services, a real-time API [8], and a math library. As architectural model, we use information such as data types and instruction set (sizes and number of execution cycles). Also program and data memory allocation aspects are considered, including allocation rules of dynamic frame and reserved memory. When timer and scheduling services are required, the overhead added in the system should be specified. In the same way, the real-time API and the math library were characterized, including resource usage information.

The estimation is evaluated for a key use case of the application, as proposed in [7]. Following this approach, the use cases that are critical to meet the application requirements are identified. After, key scenarios for these use cases are selected and represented using interaction diagrams. The estimates are computed for each scenario, and best-case and worst-case analyses can be performed.

The estimation process starts with a static analysis of the application model, which does not require simulation. From the class diagrams, structural and static information, like memory required by classes are extracted. Moreover, access methods and number and types of parameters are also identified. Furthermore,
behavioral information is extracted from the interaction diagrams, including the number of method calls and of conditional and loop operations.

From the interaction diagrams analysis, a pseudo-trace is generated that contains symbolic instructions like “getField”, “getMatrixContent”, “interactionExecution” and “loadParameterByValue”. The analysis result is mapped into the platform model, in order to obtain the estimates of system properties.

The information provided by the platform memory model is used to determine the required memory size. Information about fields in classes, together with their multiplicity, is used to compute the data memory. Moreover, the worst-case scenario of nested invocation methods is identified, and the memory volume of allocated methods is computed. The sizes of instructions and method declarations are added to estimate the program memory size, using the information about the instruction set and program memory allocation model specified in the platform. Then, this result is added to the data memory to determine the total memory size.

From the pseudo-trace and the performance and energy costs associated with the symbolic instructions, the performance and energy estimates can be computed. As there is no knowledge about the algorithms executed by the objects, just collaborations between objects and information on conditional and iteration executions are used. However, it is important to highlight that the behaviors that are most relevant to the performance estimation are captured by this methodology.

4 Experiments

In order to illustrate a design space exploration scenario, alternative solutions for a wheelchair control system were developed. For each solution, the application model was mapped into the femtoJava platform [9], based on a Java microcontroller. Afterwards, these solutions are compared using the SPEU estimations as a guide.

This experiment focuses on the wheelchair movement control, following an approach based on use cases, only the key use case and its main scenarios are evaluated. We illustrate our approach with the Movement Actuating scenario. Two different models of this scenario were developed and are illustrated in Fig 1 and Fig 2.

![Fig. 1. First solution: Sequence for Movement Actuating](image-url)
Fig. 2. Second solution: Sequence for Movement Actuating

The SPEU estimates for the two models are compared to the results obtained through a cycle-accurate simulation of the system implementation using the CACO-PS simulator [10]. The Java byte codes of the application are obtained using SASHIMI [9], the femtoJava platform synthesis tool. The results for the first and second solutions are depicted in Table 1, where it can be observed a large variation between the performance and energy for each solution. The second solution presents better results, an improvement of 66.01% in its performance and of 65.93% in its energy consumption for the best-case analysis. Similar improvements can be also observed for the worst-case analysis. The best results in the second solution are mainly due to the use of a different concurrency model, in which the scheduler needs to control only one periodic task and the system is idle for more cycles.

Table 1. Design exploration results for the alternative solutions

<table>
<thead>
<tr>
<th>Property</th>
<th>First Solution</th>
<th>Second Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPEU</td>
<td>Exact</td>
</tr>
<tr>
<td>Data memory</td>
<td>350</td>
<td>356</td>
</tr>
<tr>
<td>Program Memory</td>
<td>4,990</td>
<td>5,231</td>
</tr>
<tr>
<td>Performance best-case</td>
<td>20,732</td>
<td>21,089</td>
</tr>
<tr>
<td>Performance worst-case</td>
<td>27,068</td>
<td>27,856</td>
</tr>
<tr>
<td>Energy best-case</td>
<td>29,109,084</td>
<td>29,773,764</td>
</tr>
<tr>
<td>Energy worst-case</td>
<td>38,218,589</td>
<td>39,383,757</td>
</tr>
</tbody>
</table>

As shown in Table 1, estimates for data memory are almost exact, while for the program memory errors of 4.6% are achieved. Performance estimates present an error of at most 7.7%, and energy consumption estimates are smaller than 8.7%.

These results demonstrate that at a very high abstraction level it is already possible to explore the design space according to the system requirements, since alternative modeling solutions present different estimated results that are highly correlated to the results obtained in the real implementation. Very good estimates are obtained because the implementation reuses software components – scheduler and timer services, a real-time API, and a math library – that have been previously characterized for the hardware platform (the femtoJava processor).
5 Conclusions and Future Work

This paper proposes an approach for embedded software design space exploration using an UML-based estimation tool. To support this approach, an estimation methodology has been implemented, which allows the evaluation of models while exploring the design space, in order to find a model that better fulfills the application requirements. As expected, knowledge on the costs of reused components allows considerable improvements in the estimates.

Results have confirmed the hypothesis that it is possible to explore the design space based on modeling decisions taken at high levels of abstraction. Selecting the best modeling approach already at the UML level might result in considerable gains in terms of memory, performance, and energy consumption.

Future work will address the study of MDA in order to improve the mapping of models. Moreover, an approach for better representing the platform and its services must be defined to facilitate the system modeling. Efforts to completely automate the estimation methodology are required to make it more usable.

References