AN EMBEDDED SW DESIGN EXPLORATION APPROACH BASED ON UML ESTIMATION TOOLS

Marcio F. da S. Oliveira¹, Lisane B. de Brisolará¹, Luigi Carro^{1,2}, Flávio R. Wagner¹

¹*Computer Science Institute - Federal University of Rio Grande do Sul, PO Box 15064, Porto Alegre, Brazil;*

²*Electrical Engineering Department - Federal University of Rio Grande do Sul, Av. Osvaldo Aranha, 103, Porto Alegre, Brazil*

Abstract: Software engineers, when modeling an application using object-oriented concepts and the UML language, do not have an idea of the impact of their modeling decisions on issues such as performance, energy, and memory footprint for a given embedded platform. However, these are critical characteristics for embedded systems and should be taken into account in early design stages. In our approach, estimation of data and program memory, performance and energy are obtained 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. A case study is used to demonstrate our approach, in which an application is modeled in different ways and the alternative solutions are compared using the high-level estimations. Experimental results are presented and demonstrate the effectiveness of the estimates in an early design space exploration.

Keywords: Embedded Software, UML, Estimation, Design Space Exploration

1. INTRODUCTION

Advances in technology increase the functionality in an embedded system. At the same time, the life cycle of embedded products becomes increasingly tighter. In this scenario, productivity and quality are

simultaneously required in embedded systems design 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.

In this context, the Unified Modeling Language (UML) has gained in popularity as a tool for specification and design of embedded systems. In [2], one can find several efforts that describe the use of UML during the different phases of an embedded system design process.

Beyond the rising of abstraction level, platform-based design is required to improve the design productivity [3]. In platform-based design, design derivatives are mainly configured by software, and software development is where most of the design time is spent. But the quality of the software development also directly impacts the system properties (performance, energy consumption, memory footprint, etc.).

Nowadays, the software designer writes the application code and relies on a compiler to optimize it. Compiler code optimizations have been traditionally oriented to improve performance. However, the optimization goals for embedded systems are dominated not only by performance, but memory accesses and energy consumption are equally important [4].

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 performance, energy consumption and memory footprint. It also shows that, by using our estimation tool from UML, the designer can really explore the system software architecture, by exploring different solutions at early stages, already at the UML level. The experiment shows that the designer can obtain a reasonable amount of certitude on the best direction to be followed in order to optimize the design and meet system requirements, when comparing different modeling approaches.

This paper is organized as follows. Section 2 gives an overview of related work. Section 3 introduces our proposal for embedded software design exploration, including the adopted modeling approach and the proposed

estimation methodology. Section 4 presents the experiment that consists of an example of design space exploration scenario, comparing alternative solutions. Section 5 draws main conclusions and discusses future work.

2. RELATED WORK

Bernardi et al. [5] 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.

Theelen et al. [6] propose an UML profile and a methodology for performance modeling of UML-based systems. The UML models are translated into an executable model specified in 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 [7] 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.

Chen et al. [8] proposed a design methodology that uses an UML Platform profile to model both application and platform. In this methodology, the analysis and synthesis tools available in the Metropolis framework are used to support design space exploration. This work, however, presents a very intrusive methodology, forcing the designer to use a very particular modeling approach. It also relies on simulation to obtain performance estimations.

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. OUR PROPOSAL FOR EMBEDDED SOFTWARE DESIGN EXPLORATION

We propose an UML-based estimation approach, which 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. An estimation tool called SPEU has been developed to implement this approach. 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. Both the modeling and estimation approaches are described in this section.

3.1 Modeling approach

We chose UML as the modeling language for several reasons: (i) UML is a recognized standard that is becoming very popular, especially in industry; (ii) UML offers extension mechanisms; and (iii) UML does not depend on any particular methodology, so that one can define his/her own methodology for modeling and design space exploration.

Simple rules are imposed to the designer in order to allow automatic estimation of models. Basically, Class and Sequence diagrams are used in the application model. Structural information can be specified in the class diagram, such as multiplicity for vectors. The behavioral information is captured from the Sequence diagrams, in which we can specify interaction, conditional execution, and dependences 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 contents, like vector or matrix positions. A distinction between these methods is important because they have different costs.

The platform model could contain information about processor architecture, API, operating system, application components, and device drivers. Currently, we use only architectural aspects, 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.

3.2 Estimation approach

Our approach allows the estimation of performance, energy consumption, and data and program memory size. These system properties are the basis for other requirement analyses, like quality of service (QoS) assessment about the application in communication, schedulability, and resource usage. The approach is based on the mapping of the application model into a platform model.

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, including attributes and methods, are extracted. Moreover, access methods and number and types of parameters are also identified. Furthermore, behavioral information is extracted from the Sequence Diagrams, including the number of method calls, conditional and loop operations. A pseudo-trace is generated that contains symbolic instructions like “getField”, “getMatrixContent”, “interactionExecution”, and “loadParameterByValue”.

The result is mapped into the platform, 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 from 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. EXPERIMENT

In order to illustrate a design space exploration scenario, alternative solutions for Sokoban, a popular logic game, 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.

4.1 Alternative application models

Two alternative solutions have been modeled. However, the second one was designed to improve performance. The scenarios of the second solution use the same algorithms as the first one. Therefore, only design decisions that impact the collaborations between objects are evaluated, thus enhancing the design space exploration at this level. Figure 2 illustrates the Class Diagrams for the first solution.

Figure 1. First solution's Class Diagram.

The second solution was designed to improve performance, through the removal of the encapsulation and union of the classes *Sokoban* and *Game*. With this modification, the objects can interact directly with each other, while in the first solution the *Sokoban* class concentrated the interactions in order to encapsulate the classes like components. Therefore, the second solution has less nested invocation methods than the first one and does not

have delegation methods, thus requiring less memory and fewer cycles to be executed. Figures 2 and 3 illustrate the sequence diagrams for the *Move Player* scenario for the first and second solutions, respectively.

Figure 2. First solution's Move Player scenario.

Figure 3. Second solution's Move Player scenario.

4.2 System design exploration results

The SPEU estimates for the two models are compared against the results obtained through a cycle-accurate simulation of the system implementation using the CACO-PS power and performance simulator [10]. The hardware system and the Java byte codes of the application are obtained using

SASHIMI [9], the femtoJava platform synthesis tool. Results for the first and second solutions are depicted in Table 1.

The femtoJava platform does not have dynamic object allocation. Therefore, the data memory allocated only for class fields is the same in the estimated and in the exact values. However, the frames of method executions are dynamically allocated, thus causing a small error in the total data memory estimation. The estimated value represents the minimum data memory required for the application execution. The larger errors for the program memory were already expected, since information about the implementation of the methods is not available.

Table 1. Design exploration results for the alternative solutions.

Property	First Solution			Second Solution		
	SPEU	Exact	Error	SPEU	Exact	Error
Data memory	657	663	-0.90	645	648	-0.46
Program Memory	383	1,094	-61.86	141	1,044	-86.5
Performance best-case	42,784	113,999	-62.47	35,673	107,207	-66.73
Energy best-case	38,145,635	89,343,862	-57.31	32,354,787	83,744,347	-61.37

In the same way, the performance and energy estimated by SPEU present errors of 62,47% and 57,31% respectively, with regard to the exact values for the first solution. The SPEU estimates for the second solution present approximately the same error as the estimates for the first model. The resulting errors can be easily explained, since at the UML level there is a lack of concrete information about the actual implementation of the algorithms. However, this absolute error result does not preclude a design exploration, where a relative trend in the cost of alternative modeling solutions is more important than absolute estimates.

We can observe in Table 1 that the second solution has an estimated improvement of 16.63% in its performance. The minimum data memory required to execute the application is smaller in the second solution than in the first one. Also the estimated energy consumption was reduced in 15.18% when compared to the first solution. The program memory size was also reduced, since classes and delegation methods were removed in the second solution.

When the final implementations are compared, the second modeling solution presents an improvement of 6.06% in its performance. This smaller value, when compared to the estimated one (16.63%), is due to the error resulting from estimations at such high level of abstraction. Whenever static information is the only one available, like in [11], relatively large errors in the performance analysis of embedded software might be expected.

Nevertheless, using the relative errors one can still explore the design space in an effective manner.

However, the errors compared in Table 2 show that the estimations follow the same tendency as the exact values. The experiment showed that the second solution is really more efficient than the first one, as the estimation had indicated.

Table 2. Relative errors of the solutions (%).

Gain (%)		
Property	SPEU	Exact
Data memory	-0.90	-0.46
Program Memory	-61.86	-86.5
Performance best-case	-62.47	-66.73
Energy best-case	-57.31	-61.37

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 correlated to the results obtained in the final implementation. Therefore, the designer can explore alternative UML modeling solutions in such a way that the final synthesized application obtains improvements in performance, energy consumption, or memory footprint, or is more amenable to fulfill given system requirements.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an approach for embedded software design exploration based on UML estimation tools. To support this approach, an estimation methodology has been also proposed, which allows the evaluation of models while exploring the design space, in order to find a model that better fulfills the application requirements.

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

These facts reinforce the need for a design space exploration environment, as shown in this paper, where the designer can explore the models to obtain object-oriented applications that are as optimized as possible for embedded systems.

Future work will address the improvement of the estimations by adding information extracted from other UML diagrams and from the UML SPT

profile. Moreover, by having more detailed information about the platforms, for instance about its communication mechanisms and scheduling algorithm, we hope to increase the absolute precision of the method.

REFERENCES

1. B. Selic, Models, Software Models and UML. In: UML for Real: Design of Embedded Real-Time Systems, ch. 1, Kluwer Academic Publishers, Dordrecht, Netherlands, 2003.
2. L. Lavagno, G. Martin, B. Selic, UML for Real: Design of Embedded Real-Time Systems. Kluwer Academic Publishers, 2003.
3. D. Verkest, J. Kunkel, F. Schirrmeister, System Level Design Using C++. In: Design, Automation and Test in Europe, Paris, France, March, 2000.
4. M. Leeman et al, Automated Dynamic Memory Data Type Implentation Exploration and Optimization. In: IEEE Computer Society Annual Symposium on VLSI. Florida, February, 2003.
5. S. Bernardi, S. Donatelli, J. Merseguer, From UML Sequence Diagrams and StateCharts to Analysable Petri Net models. In: 3rd ACM International Workshop on Software and Performance, Rome, Italy, July, 2002.
6. B. D. Theelen, P.H.A. van der Putten, J.P.M. Voeten, Using the SHE Method for UML-based Performance Modeling. In: System Specification and Design Languages, ch. 12, Kluwer Academic Publishers, Dordrecht, Netherlands, 2003.
7. D.C. Petriu, C. M. Woodside, Performance analysis with UML: layered queueing models from the performance profile. In: UML for Real: Design of Embedded Real-Time Systems, ch 11 . Kluwer Academic Publishers, Dordrecht, Netherlands, 2003.
8. R. Chen, M. Sgroi, G. Martin, L. Lavagno, A. Sangiovanni-Vicentelli, J. Rabaey, Embedded System Design Using UML and Platforms. In: Forum on Specification and Design Languages, September, 2002.
9. S. Ito, L. Carro, R. Jacobi, Making Java Work for Microcontroller Applications. In: IEEE Design & Test of Computers. v. 18, n. 5, September-October, 2001.
10. A. C. Beck, J. C. B. Mattos, F. R. Wagner, L. Carro, CACO-PS: A General Purpose Cycle-Accurate Configurable Power Simulator. In: 16th Symposium on Integrated Circuits and Systems Design. São Paulo, Brazil, September, 2003.
11. S. Y. Li, S. Malik, Performance Analysis of Embedded Software Using Implicit Path Enumeration. In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v. 16, n. 12, December, 1997.