Automatic Code Generation and Solution Estimate for Object-Oriented Embedded Software

Ronaldo Rodrigues Ferreira
Instituto de Informatica
Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil
rrferreira@inf.ufrgs.br

Abstract

This work tailors an Alloy model translator to Java code and an estimate tool for physical resources optimization into a design-flow. Experimental results show distinct implementation strategies only varying data structures used in generated Java code.

Categories and Subject Descriptors C.3 [*Special-purpose and Application-based Systems*]: Real-time and Embedded Systems; D.1.2 [*Programming Techniques*]: Automatic Programming; D.2.4 [*Software Engineering*]: Software/Program Verification

General Terms Design, Languages, Measurement, Performance, Verification

Keywords Alloy, Code Generation, Design Space Exploration, Embedded Systems, Java, Modeling Languages, Software Automation

1. PROBLEM STATEMENT

The fundamental issue in embedded software development is the constrained physical resources available to software execution. Any embedded software must be carefully tuned to minimize use of hardware resources and avoid waste. Simple physical requirements such as program memory, energy and power consumption create a huge solution exploration space. Thus, automatic decision techniques for physical properties estimation and optimization, also called *design space exploration*, are mandatory to successfully choose a final system design.

Embedded system industries seek quality within their tight time-to-market windows. To enhance productivity and quality for software development, the industry is shifting its development from solely low-level coding towards high-level modeling. Model-driven approaches rely on code generators to reduce overall coding time. It is desirable that code generators using high-level models embody *formal verification* in order to assure model consistency with the application requirements.

OOPSLA '08, October 19–23, 2008, Nashville, Tennessee, USA.
ACM 978-1-60558-220-7/08/10.

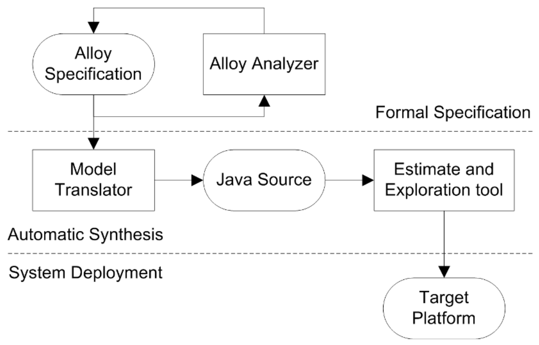

Figure 1. Proposed flow for embedded software automation

2. PROPOSED SOLUTION

This work introduces a design-flow targeting object-oriented (OO) embedded software development, which provides automatic code generation from formal high-level Alloy models plus solution estimates and exploration of physical properties. Figure 1 illustrates the proposed design-flow. *Alloy Analyzer* verification tool accompanies the Alloy language, providing an efficient state reduction algorithm before resolving the formal specification by bounded model checking. This work uses *DESEJOS* (1) tool for physical properties estimate and exploration, and adopts the *FemtoJava* (2) processor as embedded platform. This processor is a stack based Java Virtual Machine implementation that executes Java bytecodes natively. We have adopted FemtoJava and DESEJOS due to their use of Java, which makes this design-flow general enough to be adapted to others OO based platforms. This work presents the *Model Translator* implementation that translates an Alloy model into different Java implementations of the modeled application.

The transformation between Alloy and Java has several issues due to the paradigm shift from a declarative language to an object-oriented one. Alloy has relational operation semantic, does not embody explicit data structures, and does not handle some necessary constructs to program execution, such as I/O operations. The model translator performs *code synthesis* of these low-level constructs.

We have designed translation algorithms to generate *reactive systems* (RS). Often, execution of an RS is infinite. RS are a problem class where there is *finite state machine* (FSM) capturing events thrown by internal or external actors. After an event is caught, some data processing is performed accordingly to the FSM next state function, and usually the FSM control is returned to the environment. RS are widely employed in real embedded systems, mainly in control oriented applications. This work has two main contributions:

- Devising and implementing translation algorithms between Alloy and Java for control oriented RS applications;
- Proposing of an integrated design-flow for OO embedded software development automation based in a formal method.

3. EXPERIMENTAL RESULTS

Solution estimates of physical properties have been obtained for a *Drink Vending Machine* Java code generated from its Alloy model, compatible with the FemtoJava instructions set. This application is control oriented, being composed of two operations: inserting coins into the machine and buying a drink. The generated Java code contains two *collections*: one storing inserted coins and the other representing the drink storage. Figure 2 presents the results obtained for solution estimates of physical properties for five different solutions of the Vending Machine. Each solution has the same FSM; the only variation between them is the assignment of distinct data structures to the two existing collections. These five generated solutions contain the following pairs of data structures representing the coin and drink storage, respectively: #1:(*Linked List, Linked List*), #2:(*Linked List, Map*), #3:(*Linked List, Set*), #4:(*Set, Map*) and #5:(*Set, Set*). In the experiments we have used the *Javalution* (3) data structures library because it is entirely written in Java, enabling DE-

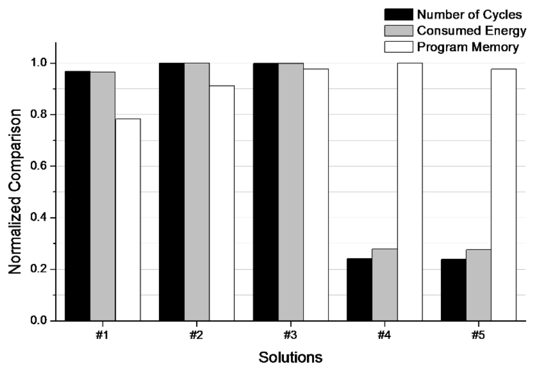

Figure 2. Physical estimates for distinct solutions

SEJOS tool to estimate physical properties of the generated solutions. These results are explained by Equation 1.

$$Energy = Power_{average} * Cycles/Frequency \quad (1)$$

Frequency and average power are constant over all solutions. These results together with Equation 1 show that only by varying used data structures, there are significant design tradeoffs between solutions, due to the variation in total number of cycles. Clearly, solutions #4 and #5 take advantage of not using linked lists, reducing algorithmic complexity from *O(n)* to *O(1)* when accessing data. This behavior is explained by the intensive elements insertion and deletion operations over object collection.

4. RELATED WORK

ForSyDe (4) is a framework for modeling and synthesizing embedded systems which uses Haskell as specification language. Although adopting a language with well defined semantics, ForSyDe lacks formal verification and design space exploration. Its main advantage is its tailored deployment of software and hardware, often called *co-design*.

In addition to integrated hardware and software co-design, Metropolis provides formal verification based on PROMELA specifications and on the SPIN model checker (5). It also provides execution trace simulation, requiring less effort and time to check system integrity than model checking, however without formal correctness. Metropolis does not offer integrated design space exploration and solution estimation.

5. FUTURE WORK

The next step is the model translator extension to support the *Synchronous Data Flow* (SDF) (6) model of computation. To do so, we need to define a SDF Alloy library and the translation algorithms to it. With SDF it will be possible to perform solution selection by means of models of computation, leveraging current solution exploration techniques.

References

[1] Mattos, J. C. B., and Carro, L. 2007. Object and Method Exploration for Embedded Systems Applications. 20th SBCCI, Brazil. ACM Press, New York. p. 318-323.

[2] Ito, S. A., Carro, L. and Jacobi, R. P. 2001. Making Java work for microcontroller applications. IEEE Design & Test of Computers, 18 (5), p. 100-110.

[3] Dautelle, J. 2007. Fully Deterministic Java. AIAA SPACE 2007 Conference and Exposition, California, p. 18-20.

[4] Sander, I. and Jantsch, A. 2004. System Modeling and Transformational Design Refinement in ForSyDe. IEEE TCAD, 23 (1), p. 17-32.

[5] Balarin, F., et.al. 2003. Metropolis: An Integrated Electronic System Design Environment. Computer, 36 (4), p. 45-52.

[6] Lee, E. A. and Messerschmitt, D. G. 1987. Synchronous Data Flow. Proceedings of the IEEE, 75 (9), p. 1235-1245.