Code Generation for Embedded Systems

Peter Sestoft Henrik Hulgaard Andrzej Wąsowski

IT-C, 3-4 June 2002

keywords: statecharts, UML, code generation, code synthesis, embedded systems, interpreters.

Background

Over the last decade IT technology has entered the market of industrial and home appliances. The number of specialized micro-processors embedded in devices already far exceeds the number of CPUs in personal computers. Their overwhelming success motivates the development of methodologies and tools supporting definition and implementation of control algorithms for embedded systems.

Development of embedded software faces somewhat different problems than standard PC programming, where memory is practically unlimited (for most applications) and computation speed is fast and ever increasing. Standard workstation compilers optimize for speed and are implemented to get short compilation times. Despite the success of multi-threaded operating systems, most of the program logic is still inherently sequential.

In contrast, embedded systems are highly concurrent and time constrained. The size of memory (both read-only ROM and writable RAM) is limited. Both memory size and CPU speed strongly influence power consumption and cost, and should be minimized.

Usage of textual languages (C/C++/assembly languages) still plays the primary role in development of embedded software. In this traditional context, design of control algorithms resembles development of standard PC software. Nevertheless, the specification of control algorithms in C-like languages with very limited support for concurrency is inconvenient and leads to cumbersome and bug-prone designs. Moreover, a significant amount of information about the system's concurrency structure is lost during design, and is not available at compile-time.

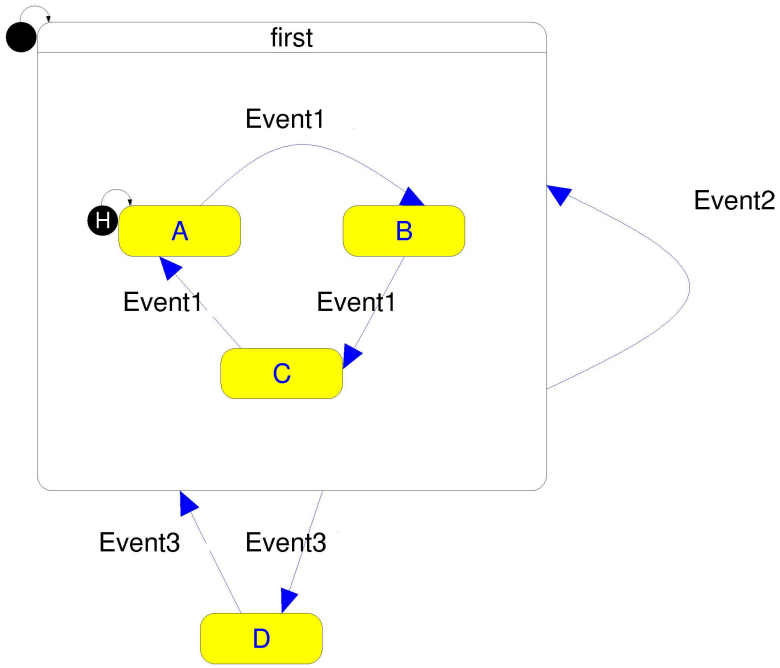

Figure 1: Sample statechart model.

Project Description

Visual design languages (*Mealy machines, Statecharts, Petri nets*) present attractive alternatives for the design of complex control algorithms. Statecharts (Fig. 1) are the most popular visual language among those, mainly because of their incorporation in the UML family of diagrams. Statecharts provide a readable and convenient way to visualize control algorithms of hierarchically structured concurrent reactive systems. The use of hierarchy increases readability and avoids size explosion.

In this project we consider automatic program synthesis for micro-controller driven systems. The source language are statecharts as used in the VisualSTATE tool (IAR Inc.) and UML state diagrams. The main focus is on the specific needs of embedded software.

The information provided by a hierarchical concurrent model gives more opportunity for an optimizing compiler than a C program with occasional calls to a branching function. Consequently a Statechart compiler has an advantage over a traditional C compiler, so it should be possible to generate fast but compact code. Additional parameters (power consumption, meeting time constraints, etc) as well as various optimization trade-offs may be taken into account as well.

Problems and Related Theory

The extensive theory of *translation* is our primary source. Traditionally, code synthesis from statecharts has been based on interpretative techniques: a runtime representation of the model is generated, which is then executed by an interpreter.

Part of our work concerns *data structures* for internal representation of models. Preserving high-level information in the data structure should help improve code compactness and speed. The code generator currently used by VisualSTATE discards hierarchy information by flattening the model, sometimes causing a state explosion. We attempt to preserve high-level information by using tree-like model representations. Our current interpreter executes such hierarchical models directly.

A model representation has static and dynamic parts, reflecting features of the model and the kinds of memory available. Memory is a scarce resource, but the read-write (RAM) part is even more limited than the read-only part (ROM). Thus the dynamic features of the model (global state, history) should be represented in a highly compact manner. When dynamic memory is not a scarce resource, it is interesting to control the trade-off between static and dynamic memory sections.

The current VisualSTATE implementation uses the same interpreter for all models. Instead, one could generate a specialized interpreter for each model. The degree of specialization may range from nothing (plain interpretation) to full *partial evaluation* (a hard-coded statechart model). We aim to experiment with the boundary between interpretation and compilation to control the size/speed trade-off.

Various state space analyses of the model, possibly based on *model-checking*, may provide significant information, which could be used to generate better code.

Current Advances and Expected Results

An extensive study of statechart languages has been made, and a formal semantics for VisualSTATE statecharts has been proposed. A toolkit for the language has been developed, which currently includes several minor tools and a highly abstract experimental model interpreter. It will soon be equipped with a code generator.

The work should result in a comprehensive analysis of synthesis techniques for statecharts, including new contributions. The form should be suitable for direct transfer to companies implementing visual tools.

We will also develop an open source code generator for VisualSTATE statecharts, covering other statechart variants as well, to understand their pros and cons.

Wider Perspective

Within the IT-C this work belongs to the project *Resource Constrained Embedded Systems* (RCES). In a wider context it fits into a chain of research efforts on providing a complete and reliable methodology for developing software:

- design of usable modeling languages;
- defining the formal semantics of design languages;
- model-checking and verification;
- code synthesis from models;
- automatic testing and test case generation.

Related Work

In the early eighties Berry (INRIA) proposed *Esterel*, a synchronous textual language for concurrent software, having many similarities with statecharts. Intensive research has been done on Esterel compilation.

In 1987 Harel (Weizmann Institute) introduced statecharts, then implemented in the Statemate tool (I-Logix). The work was mainly focused on semantics of statecharts (Harel and Pnueli) and hardware synthesis resulting in a PhD thesis by Drusinsky (1988).

More recently, attempts have been made to translate UML state diagrams to imperative and formal languages (Sekerinski at McMaster, Lilius and Paltor at Turku). In none of the cases the focus was on embedded systems with constrained resources.

IAR Inc. cooperates with verification researchers on improving code synthesis in VisualSTATE (Kristoffersen at the IT-C, Behrmann at Aalborg). A synthesis scheme has been developed that preserves hierarchical structure; it is being implemented by VisualSTATE.

The IMAGES project (Rajkumar and Clarke at Carnegie Mellon) develops a complete framework for embedded software. One of the objectives is to use verification results (in form of annotations) to improve code synthesis.