On Generating Soft Real-Time Programs for Non-Real-Time Environments

Ilankaikone Senthooran and Takuo Watanabe

Department of Computer Science, Tokyo Institute of Technology
2-12-1 Ookayama, Meguroku, Tokyo, 152-8552, Japan
senthooran@psg.cs.titech.ac.jp, takuo@acm.org

Abstract. Model based development of real-time embedded systems has actively been studied. Generating real-time code from a model is an important topic of these studies. In this paper, we briefly describe a code generation scheme for real-time programs for non-real-time environment. The generated code runs in a runtime environment without inherent real-time scheduling facilities. In our approach, models are formally written as systems of timed automata and are verified using UPPAAL model checker prior to be processed by our code generator. In order to make the generated code to meet the timing requirements in a non-real-time environment, the code generator weaves explicit timing checking code fragments in the code. We construct a simple two-wheeled robot as a case study of this approach.

Keywords: timed automata, model checking, UPPAAL, code generation, soft real-time systems.

1 Introduction

The primary motivation of this work is to provide an easy and systematic way of developing soft real-time systems. This paper presents a method for generating real-time programs running on top of non-real-time runtime environments. Overview of our method, which is an instance of model-based development, is described briefly in Fig. 1. The model is used in two ways: it is verified by the model verifier with respect to given properties and then is used as the source of the code generator. We adopt timed automata \cite{2,4} for model description and thus use UPPAAL model checker \cite{10,3} as the model verifier.

Code generation plays an important role in model-based development. We designed a code generation scheme that translates models written as a system of UPPAAL timed automata into plain Java programs. Based on this scheme, we implemented a code generator named TA2J. The generated code can be deployed to runtime environments that have no inherent real-time schedulers. The code explicitly checks the timing constraints by polling real-time clocks to meet the timing requirements specified in the model.

The rest of the paper is organized as follows. The next section briefly reviews the concepts of Timed Automata (TA) and real-time system modeling using...
2 I. Senthoooran and T. Watanabe

TA2J/TA2C (code generator)

Model in TA

Model Props in TCTL

UPPAAL (model verifier)

Generated Java/C Code

Runtime (VM, OS)

w/ explicit timing constraint checking

w/o native real-time schedulers

Fig. 1. Overview of Our Method

TAs. Section 3 presents our code generation scheme and Section 4 shows a case study — a small two-wheeled robot. Section 5 overviews related work. The final section concludes the paper.

2 Modeling Real-Time Systems Using Timed Automata

Modeling formalisms for real-time systems have to capture both quantitative and qualitative aspects of time. For modeling real-time embedded systems, UML StateCharts and Stateflow are gaining popularity these days. However, they are weak in their support for formal verification.

Several formalisms that support verification have been investigated. For example, timed extensions of Petri Nets and process algebras, real-time logics and timed automata [12]. In this work, we adopt timed automata [2] as the primary modeling formalism and use model checking for the verification purpose.

A timed automaton is an extension of a Büchi automaton enhanced with real-valued clock variables. Fig. 2 shows an example of timed automaton modeling a simple lamp with a button [3]. It has three locations named off, dim and bright that represent the state of the lamp. In this example, off corresponds to the initial state. The label press? on each edge represents an action of pressing the button. Suppose that the lamp is off. When we press the button once, the lamp is turned on and becomes dim. But if we quickly press the button twice, the lamp is turned on and becomes bright.

To model such timing dependent behaviors using timed automata, we use clock variables and clock constraints. Clock variables $x, y, \ldots$ are real-valued variables that increase continuously at the same rate with time. Clock constraints are expressions associated to edges or locations. They are defined using the abstract syntax $g ::= x < c \mid x \leq c \mid x = c \mid x \geq c \mid x > c \mid g \land g$ where $x$ is a clock variable and $c$ is a non-negative integer constant.

1 In timed automata terminology, a state in an automaton is called a location.

2 Constants in clock constraints are restricted to integers for model-checking purpose.