Available online at www.sciencedirect.com
ScienceDirect
Computer Languages, Systems & Structures 34 (2008) 153–169
COMPUTER LANGUAGES, SYSTEMS & STRUCTURES
www.elsevier.com/locate/cl

Automatic synthesis and verification of real-time embedded software for mobile and ubiquitous systems

Pao-Ann Hsiung*, Shang-Wei Lin

Department of Computer Science and Information Engineering, National Chung-Cheng University, Chiayi, Taiwan, ROC

Received 15 November 2006; accepted 1 June 2007

Abstract

Currently available application frameworks that target the automatic design of real-time embedded software are poor in integrating functional and non-functional requirements for mobile and ubiquitous systems. In this work, we present the internal architecture and design flow of a newly proposed framework called *Verifiable Embedded Real-Time Application Framework* (VERTAF), which integrates three techniques namely software component-based reuse, formal synthesis, and formal verification. Component reuse is based on a formal unified modeling language (UML) real-time embedded object model. Formal synthesis employs quasi-static and quasi-dynamic scheduling with multi-layer portable efficient code generation, which can output either real-time operating systems (RTOS)-specific application code or automatically generated real-time executive with application code. Formal verification integrates a model checker kernel from state graph manipulators (SGM), by adapting it for embedded software. The proposed architecture for VERTAF is component-based which allows plug-and-play for the scheduler and the verifier. The architecture is also easily extensible because reusable hardware and software design components can be added. Application examples developed using VERTAF demonstrate significantly reduced relative design effort as compared to design without VERTAF, which also shows how high-level reuse of software components combined with automatic synthesis and verification increases design productivity.

Keywords: Application framework; Code generation; Real-time embedded software; Formal synthesis; Formal verification; Scheduling; Software components; UML modeling

1. Introduction

With the proliferation of embedded mobile and ubiquitous systems in all aspects of human life, we are making greater demands on these systems, including more complex functionalities such as pervasive computing, mobile computing, embedded computing, and real-time computing. Currently, the design of real-time embedded software is supported partially by modelers, code generators, analyzers, schedulers, and frameworks [1–21]. Nevertheless, the technology for a completely integrated design and verification environment is still relatively immature. Furthermore, the methodologies for design and for verification are also poorly integrated relying mainly on the experiences of embedded software engineers. Motivated by the above status-quo, this work demonstrates how the integration of software engineering

* Corresponding author. Tel.: +886 5 2720411; fax: +886 5 2720859.
E-mail addresses: hpa@computer.org, pahsiung@cs.ccu.edu.tw (P.-A. Hsiung).

1477-8424/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cl.2007.06.002

techniques such as software component reuse, formal software synthesis techniques such as scheduling and code generation, and formal verification technique such as model checking can be realized in the form of an integrated design environment targeted at the acceleration of real-time embedded software construction.

Mobile and ubiquitous systems involve the dynamic reconfiguration of applications in response to changes in their environments. Middlewares such as network layer mobility support, transport layer mobility support, traditional distributed systems applied to mobility, middleware for wireless sensor networks, context awareness-based middleware, and publish-subscribe middleware are required for efficient development of mobile and ubiquitous applications. A user can develop an application using such middlewares, however, it can sometimes be too tedious and complex to consider all the different possible environments and application features. Examples of environments include office and domestic spaces, educational and healthcare institutions and in general urban and rural environments. Examples of applications include domestic and industrial security applications, education and learning type applications, healthcare applications, traffic management, commercial advertising, games and arts, and more extreme applications, such as applications for rescue operations and the military. Given such complex combinations of environments and applications, one would desire a higher level of reuse than that allowed by object-oriented design and middlewares. We are thus proposing an integrated design framework that allows such higher level of reuse.

Several issues are encountered in the development of an integrated design framework. First and foremost, we need to decide upon an architecture for the framework. Since our goal is to integrate reuse, synthesis, and verification, we need to have greater control on how the final generated application will be structured, thus we have chosen to implement it as an object-oriented application framework [22], which is a "semi-complete" application, where users fill in application specific objects and functionalities. A major feature is "inversion of control", that is the framework decides on the control flow of the generated application, rather than the designer. Other issues encountered in architecting an application framework for real-time embedded software are as follows.

1. To allow software component reuse, how do we define the syntax and semantics of a reusable component? How can a designer uniformly and guidedly specify the requirements of a system to be designed? How can the existing reusable components with the user-specified components be integrated into a feasible working system?
2. What is the control-data flow of the automatic design and verification process? When do we verify and when do we schedule?
3. What kinds of model can be used for each design phase, such as scheduling and verification?
4. What methods are to be used for scheduling and for verification? How do we automate the process? What kinds of abstraction are to be employed when system complexity is beyond our handling capabilities?
5. How do we generate portable code that not only crosses real-time operating systems (RTOS) but also hardware platforms. What is the structure of the generated code?

Briefly, our solutions to the above issues can be summarized as follows.

1. *Software component reuse and integration*: A subset of the Unified Modeling Language (UML) [23] is used with minimal restrictions for automatic design and analysis. Precise syntax and formal semantics are associated with each kind of UML diagram. Guidelines are provided so that requirement specifications are more error-free and synthesizable.
2. *Control flow*: A specific control flow is embedded within the framework, where scheduling is first performed and then verification because the complexity of verification can be greatly reduced after scheduling [4].
3. *System models*: For scheduling, we use variants of Petri nets (PN) [6,7] and for verification, we use Extended Timed Automata (ETA) [7,24], both of which are automatically generated from user-specified UML models that follow our restrictions and guidelines.
4. *Design automation*: For synthesis, we employ quasi-static and quasi-dynamic scheduling methods [6,7] that generate program schedules for a single processor. For verification, we employ symbolic model checking [25–27] that generates a counterexample in the original user-specified UML models whenever verification fails for a system under design. The whole design process is automated through the automatic generation of respective input models, invocation of appropriate scheduling and verification kernels, and generating reports or useful diagnostics. For handling complexity, abstraction is inevitable, thus we apply model-based, architecture-based, and function-based abstractions during verification.