Verifiable Embedded Real-Time Application Framework

Shang-Wei Lin[†], Kuo-Cheng Chiang, and Pao-Ann Hsiung[‡]
Department of Computer Science and Information Engineering
National Chung Cheng University
[†]linsw@cs.ccu.edu.tw, [‡]hpa@computer.org

Abstract

Currently available application frameworks that target at the automatic design of real-time embedded software are poor in integrating functional and non-functional requirements for real-time embedded 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 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 RTOS-specific application code or automatically-generated real-time executive with application code. Formal verification integrates a model checker kernel from SGM, by adapting it for embedded software. 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 increase design productivity.*

1. Introduction

With the advancement of technology, the demand for new and complicated features of embedded systems are getting more and more strong, which makes the correctness of embedded systems very difficult to verify. However, guaranteeing the correctness of embedded systems is getting more and more important because embedded systems are becoming more and more pervasive. Currently, the design of real-time embedded software is supported partially by modelers, code generators, analyzers, schedulers, and frameworks [2, 6, 8, 9, 10, 11, 12, 13, 15, 14, 16, 5, 21, 22, 23, 24, 25, 28, 29]. 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.

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

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 [18], 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) [20] 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 [9].

3. System Models: For scheduling, we use variants of Petri Nets (PN) [11, 12] and for verification, we use Extended Timed Automata (ETA) [1, 12], 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 [11, 12] that generate program schedules for a single processor. For verification, we employ symbolic model checking [3, 4, 19] 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.

5. Portable Efficient Multi-Layered Code: For portability, a multi-layered approach is adopted in code generation. To account for performance degradation due to multiple layers, system-specific optimization and flattening are then applied to the portable code. System dependent and independent parts of the code are distinctly segregated for this purpose.

In summary, this work illustrates how an application framework may integrate all the above proposed design and verification solutions. Our implementation has resulted in a Verifiable Embedded Real-Time Application Framework (VERTAF) whose features include formal modeling of real-time embedded systems through well-defined UML semantics, formal synthesis that guarantees satisfaction of temporal as well as spatial constraints, formal verification that checks if a system satisfies user-given properties or system-defined generic properties, and code generation that produces efficient portable code.

This paper is organized as follows. The design and verification flow are described in Section 2. Section 3 presents an application result. Section 4 gives the conclusion of this paper.

2. Design and Verification Flow in VERTAF

Before going into the component-based architecture of VERTAF, we first introduce the design and verification flow. As shown in Figure 1, VERTAF provides solutions to the various issues introduced in Section 1.

In Figure 1, the control and data flows of VERTAF are represented by solid and dotted arrows, respectively. Software synthesis is defined as a two-phase process: a machine-independent software construction phase and a machine-dependent software implementation phase. This separation helps us to plug-in different target languages, middleware, real-time operating systems, and hardware device configurations. We call the two phases as front-end and back-end phases. The front-end phase is further divided into three sub-phases, namely UML modeling phase, real-time embedded software scheduling phase, and formal verification phase. There are two sub-phases in the back-end phase, namely component mapping phase and code generation phase. We will now present the details of each phase in the rest of this section illustrated by a running example called Entrance Guard System with Mobile and Ubiquitous Control (EGSMUC). EGSMUC is a real-time embedded system that controls any entrance with a programmable electronic lock installed. Two ways of control accesses are allowed: (a) registered users can be authenticated locally at the entrance itself, and (b) guest users may obtain a remote authentication through master acknowledgment. Here, a master could be the owner of the building to which the entrance system is protecting and he or she can have mobile and ubiquitous control access to EGSMUC. The master can grant entry access to the guest user irrespective of how he or she is connected to EGSMUC (mobile access) and also irrespective of where he or she is located (ubiquitous access). We will model EGSMUC and VERTAF will automatically synthesize and verify the code for the system.

2.1 UML Modeling

UML [20] is one of the most popular modeling and design languages in the industry. After scrutiny

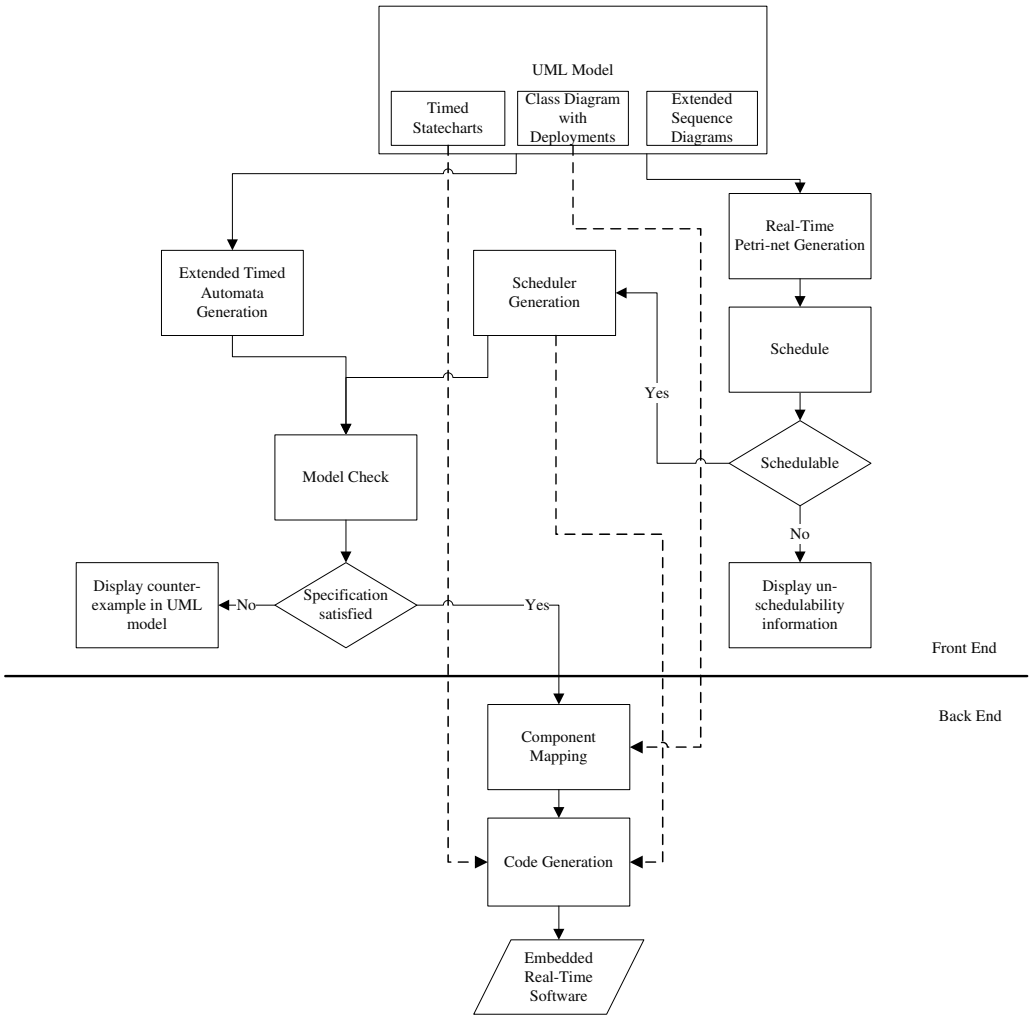

Figure 1. Design and Verification Flow of VERTAF

of all diagrams in UML, in order to minimize the information redundancy we have chosen three diagrams for a user to input as system specification models, namely class diagram, sequence diagram, and statechart. UML is a generic language and its specializations are always required for targeting at any specific application domain. In VERTAF, the three UML diagrams are both restricted as well as enhanced along with guidelines for designers to follow in specifying synthesizable and verifiable system models (just as synthesizable HDL code for hardware designs).

The three UML diagrams extended for real-time embedded software specification are as follows.

- *Class Diagrams with Deployment*: A deployment relation is used for specifying a hardware object on which a software object is deployed. There are two types of methods, namely event-triggered and time-triggered that are used to model real-time behavior.
- *Timed Statecharts*: UML statecharts are extended with real-time clocks that can be reset and values checked as state transition triggers.
- *Extended Sequence Diagrams*: UML sequence diagrams are extended with control structures such as concurrency, conflict, and composition, which aid in formalizing their semantics and in mapping them to formal Petri net models that are used for scheduling.

For our running EGSMUC example, the system class diagram with deployment is shown in Figure 2, a timed statechart for the system controller class is shown in Figure 3, and an extended sequence diagram for one of the use cases dealing with guest entry and master acknowledgment is shown in Figure 4.

UML is well-known for its informal and general-purpose semantics. The enhancements described above are an effort at formalizing semantics preciseness such that there is little ambiguity in user-specified

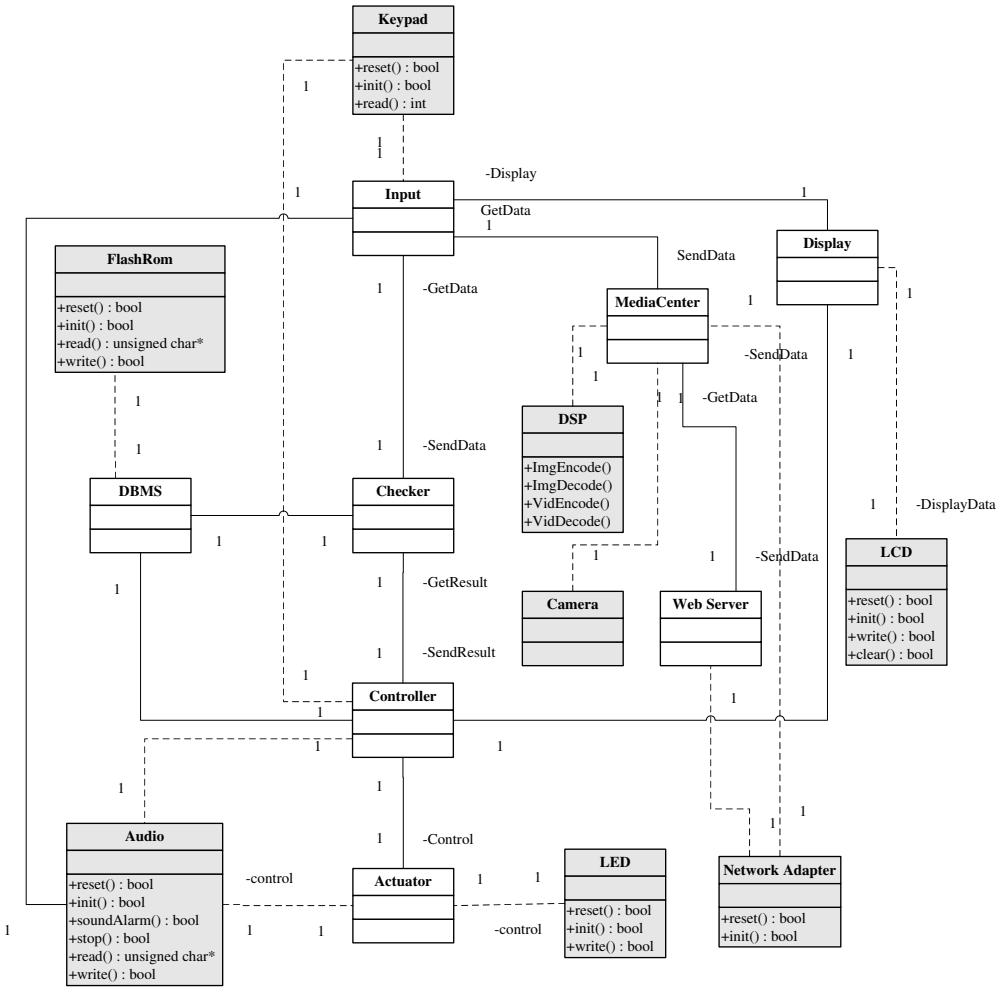

Figure 2. Class Diagram with Deployment for EGSMUC

models that are input to VERTAF. Furthermore, design guidelines are provided to a user such that the goal of correct-by-construction can be achieved.

The set of UML diagrams input by a user, including a class diagram with deployments, a timed statechart corresponding to each class, and a set of extended sequence diagrams, constitutes the requirements for the real-time embedded software to be designed and verified by VERTAF.

2.2 Real-Time Embedded Software Scheduling

There are two issues in real-time embedded software scheduling, namely how are memory constraints satisfied and how are temporal specifications such as deadlines satisfied. Based on whether the system under design has an RTOS specified or not, two different scheduling algorithms are applied to solve the above two issues.

- *Without RTOS*: *Quasi-dynamic scheduling* (QDS) [11, 12] is applied, which requires *Real-Time Petri Nets* (RTPN) as system specification models. QDS prepares the system to be generated as a single real-time executive kernel with a scheduler.

- *With RTOS*: *Extended quasi-static scheduling* (EQSS) [26] with real-time scheduling [17] is applied, which requires *Complex Choice Petri Nets* (CCPN) and set of independent real-time tasks as system specification models, respectively. EQSS prepares the system to be generated as a set of multiple threads that can be scheduled and dispatched by a supported RTOS such as MicroC/OS II or ARM Linux.

In order to apply the above scheduling algorithms, we need to map the user-specified UML models into Petri nets, RTPN or CCPN. By applying the mapping procedure, all user-specified sequence diagrams are translated and combined into a compact set of Petri nets. All kinds of temporal constraints that appear in the sequence diagrams are translated into guard constraints on arcs in the generated Petri nets. This set of RTPN or CCPN is then input to QDS or EQSS, respectively, for scheduling. Details on the scheduling

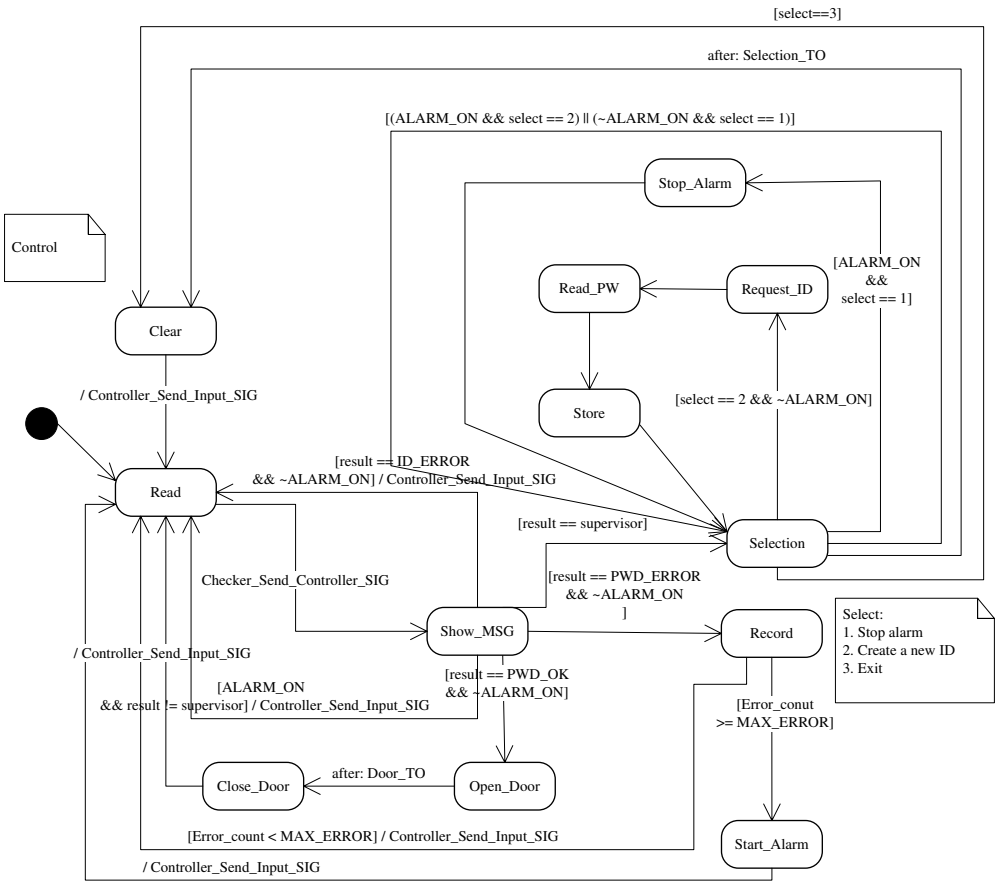

Figure 3. Timed Statechart for Controller in EGSMUC

procedures can be found in [11, 12], and [26].

For systems without RTOS, we need to automatically generate a scheduler that controls the system according to the set of transition sequences generated by QDS. In VERTAF, a scheduler is constructed as a separate class that observes and controls the status of each object in the system. Temporal constraints are monitored by the scheduler class using a global clock. Further, for verification purposes, an extended timed automaton is also generated by following the set of transition sequences.

For our running EGSMUC example, a single Petri net is generated from the user-specified set of statecharts, which is then scheduled using QDS. In this example, scheduling is required only for the timers associated with the actuator, the controller, and the input object. After QDS, we found that EGSMUC is schedulable.

2.3 Formal Verification

VERTAF employs the popular model checking paradigm for formal verification of real-time embedded software. In VERTAF, formal ETA models are generated automatically from user-specified UML models by a flattening scheme that transforms each statechart into a set of one or more ETA, which are merged, along with the scheduler ETA generated in the scheduling phase, into a state-graph. The verification kernel used in VERTAF is adapted from *State Graph Manipulators* (SGM) [28], which is a high-level model checker for real-time systems that operate on state-graph representations of system behavior through manipulators, including a state-graph merger, several state-space reduction techniques, a dead state checker, and a TCTL model checker. There are two classes of system properties that can be verified in VERTAF: (1) system-defined properties including dead states, deadlocks, livelocks, and syntactical errors, and (2) user-defined properties specified in the *Object Constraint Language* (OCL) as defined by OMG in its UML specifications. All of these properties are automatically translated into TCTL specifications for verification by SGM.

Automation in formal verification of user-specified UML models of real-time embedded software is achieved in VERTAF by the following implementation mechanisms.

1. User-specified timed statecharts are automatically mapped to a set of ETA.

2. User-specified extended sequence diagrams are automatically mapped to a set of Petri nets that

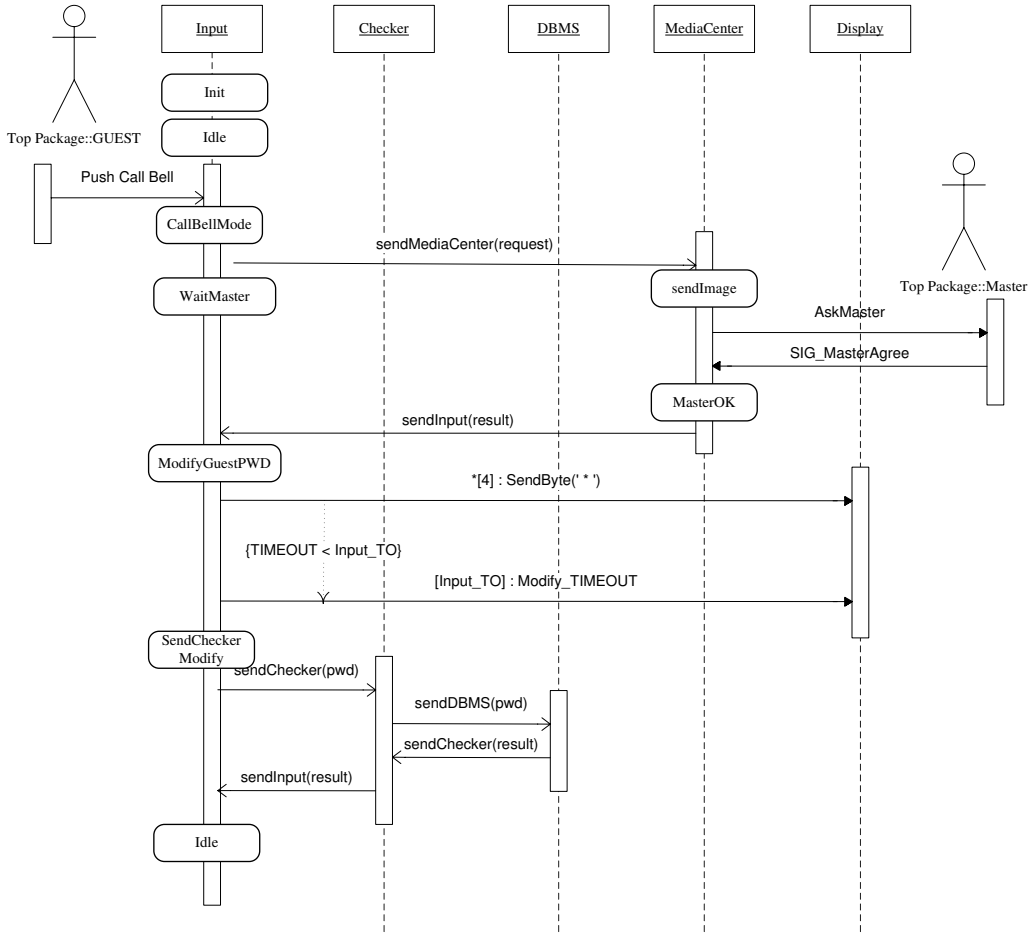

Figure 4. An Extended Sequence Diagram for EGSMUC

are scheduled and then a scheduler ETA is automatically generated.

3. Using the state-graph merge manipulator in SGM, all the ETA resulting from the above two steps are merged into a single state-graph representing the global system behavior.

4. User-specified OCL properties and system-defined properties are automatically translated into TCTL specification formulas.

5. The system state-graph and the TCTL formulas obtained in the previous two steps are then input to SGM for model checking.

6. When a property is not satisfied, SGM generates a counterexample, which is then automatically translated into a UML sequence diagram representing an erratic trace behavior of the system.

Design complexity is a major issue in formal verification, which leads to unmanageable and exponentially large state-spaces. Both engineering paradigms and scientific techniques are applied in VERTAF to handle the state-space size explosion issue. The applied techniques include *Model Construction Guidelines*, *Architectural Abstractions* [7, 25, 27, 30], *Functional Abstractions*, and State-Space Reductions. These abstraction techniques are applied to a user-specified UML model as follows. While constructing the UML models, users not following the guidelines are warned of the possible intricacies. Upon completion of model construction, first Petri net models are generated, which are then scheduled to produce feasible system schedules that are represented by a scheduler ETA. Then, for each ETA generated from the statecharts, its assumptions and guarantees are generated. The guarantees of an ETA are verified by first merging the ETA with functional abstractions of the other ETA in the system and then reducing the state-spaces of the merged state-graph using SGM reduction manipulators. We can see that not only is verification automated but abstraction techniques such as AGR and state-space reductions are also automatically performed, which makes VERTAF scalable to large applications.

For our running EGSMUC example, the ETA for each statechart were generated and then merged with the scheduler ETA. There are eight other ETA in this system example. All ETA were input to SGM and AGR was applied. Reduction techniques were then applied to each state-graph obtained from AGR. OCL

constraints were then translated into TCTL and verified by the SGM model checker kernel.

2.4 Component Mapping

This is the first phase in the back-end design of VERTAF and starts to be more hardware dependent. All hardware classes specified in the deployments of the class diagram are those supported by VERTAF and thus belong to some existing class libraries. The component mapping phase then becomes simply the configuration of the hardware system and operating system through the automatic generation of configuration files, make files, header files, and dependency files. The corresponding hardware class API will be linked in during compilation.

The main issue in this phase occurs when a software class is not deployed on any hardware component or not deployed on any specific hardware device type, for example the type of microcontroller to be used is not specified. Currently, VERTAF adopts an interactive approach whereby the designer is warned of this lack of information and he/she is requested to choose from a list of available compatible device types for the deployment. An automatic solution to this issue is not feasible because estimates are not easy without further information about the non-deployed software classes.

Another issue in this phase is the possible conflicts among hardware devices specified in a class diagram such as interrupts, memory address ranges, I/O ports, and bus-related characteristics such as device priorities. Users are also warned in case of such conflicts.

For our running EGSMUC example, all software classes in the class diagram given in Figure 2 are deployed on one or more hardware or software classes supported by VERTAF.

2.5 Code Generation

There are basically three issues in this phase including hardware portability, software portability, and temporal correctness. We adopt a multi-tier approach for code generation: an operating system layer, a middleware layer, and an application with scheduler layer, which solves the above three issues, respectively. Currently supported underlying hardware platforms include dual core ARM-DSP based, single core ARM, StrongARM, or 8051 based, and Lego RCX-based Mindstorm systems. For hardware abstraction, VERTAF supports MicroHAL and the embedded version of POSIX. For operating systems, VERTAF supports MontaVista Linux, MicroC/OS, Embedded Linux, and eCOS. For middleware, VERTAF is currently based on the Quantum Framework [21]. For scheduler, VERTAF creates a custom ActiveObject according to the Quantum API. Included in the scheduler is a temporal monitor that checks if any temporal constraints are violated. A sample configuration is shown in Figure 5, where the multi-tier approach decouples application code from the operating system through the middleware and from the hardware platform through the operating system layer.

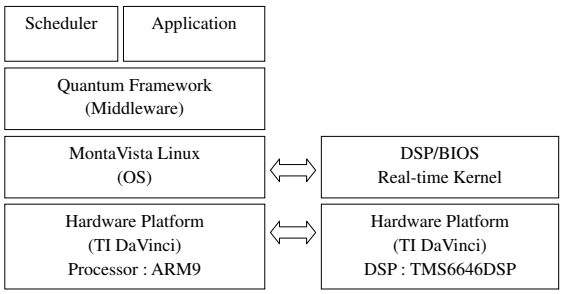

Figure 5. Multi-Tier Code Architecture

Each ETA that is generated either from UML statecharts or from the scheduled Petri nets (sequence diagrams) is implemented as an ActiveObject in the Quantum Framework. The user-defined classes along with data and methods are incorporated into the corresponding ActiveObject. The final program is a set of concurrent threads, one of which is a scheduler that can control the other objects by sending messages to them after observing their states. For systems without an OS, the scheduler also takes the role of a real-time executive kernel.

For our running example, the final application code consisted of 7 activeobjects derived from the statecharts and 1 activeobject representing the scheduler. Makefiles were generated for linking in the API of the 8 hardware classes and configuration files were generated for the ARM-DSP dual microprocessor platform called DaVinci from Texas Instruments with MontaVista Linux as its operating system on the ARM processor and DSP/BIOS real-time kernel as the operating system on the DSP TMS6646DSP processor. There were totally 2,754 lines of C code for the full EGSMUC system, out of which the system designers had to write only around 170 lines of C code, which is only 6.2% of the full system code.

3. Application Results

For the running example EGSMUC, we now analyze why VERTAF is capable of generating a significant part of the system implementation code, thus alleviating the designer from the tedious and error-prone task of manual coding. Due to its application framework architecture, VERTAF supports software components that are commonly found in mobile, ubiquitous, real-time, and embedded application domains. We classify the components supported by VERTAF into four classes, namely, storage and I/O devices, communication interfaces, multimedia processing units, and control and management interfaces.

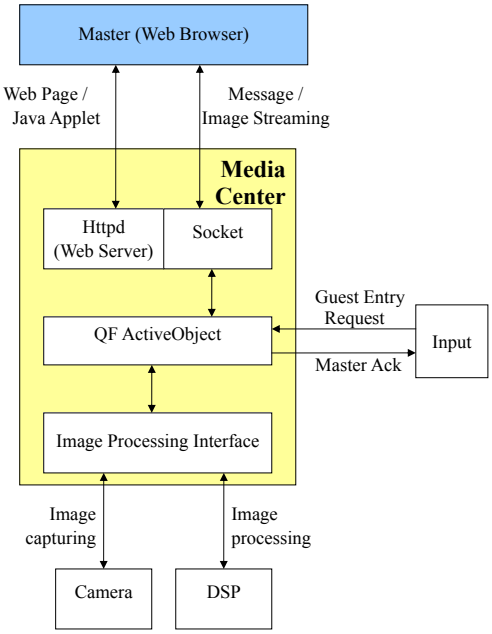

Figure 6. Code Structure for Media Center in EGSMUC

To implement mobile and ubiquitous control access in a real-time embedded system, a user normally, without VERTAF, would have to install a web server, write multimedia processing code, write network code, and integrate everything together, along with application-specific context awareness or publish-subscribe middlewares. With VERTAF, most of these tedious work are not required as long as the user configures the correct components from the framework for use in his or her application.

Figure 4 gave the sequence diagram that a user needs to specify in order for VERTAF to generate corresponding code. The architecture of the code generated by VERTAF is shown in Figure 6, where QF ActiveObject is an active object from the Quantum Framework. The code consists of three parts, namely a web server, a QF activeobject, and an image processing interface. The web server allows a master to connect to EGSMUC using a web browser that can run Java applets. The applet opens a socket connection between the media center and the client machine of the master. The image of the guest requesting entrance is captured and processed through the image processing interface. When a master acknowledges, the guest is notified through the input class. The control and data flows of the media center are automatically generated by VERTAF and the user has to merely specify the sequence diagrams as shown in Figure 4 and deploy the related classes to hardware or software components in the class diagram as shown in Figure 2. This is exactly the reason why VERTAF can save a lot of coding and design efforts.

Figure 7 shows the entire flow from the designing phase to the implementation of the EGSMUC system running on the TI DaVinci board. The scheduled and verified code generated by VERTAF is then uploaded into the TI DaVinci board. The bottom of the figure shows the demo of the EGSMUC system. When a guest requests for entrance, the master can get the image of the guest captured by the camera on the TI DaVinci board via the internet with a mobile device such as a PDA to grant the permission. With the permission of the master, the guest can key in the password using the remote controller, and the door will be opened.

There were totally 16 objects in the final application generated by VERTAF, out of which the user or designer had to only model 7 classes. The remaining 9 classes included components from all the four categories as described at the start of Section 3. Empirical results obtained from comparing two different implementations of the EGSMUC system, one using VERTAF, and one without using VERTAF, showed that not only the user written code reduced to 6.2% and the number of objects reduced to 44%, but the total time required to develop the application also reduced by more than 60%. The average learning time for each designer using VERTAF was approximately 0.1 day. The experimental and empirical results all show that VERTAF is beneficial to designers of real-time embedded software with mobile and ubiquitous control access.

4. Conclusion

An object-oriented component-based application framework, called VERTAF, was proposed for the development of real-time embedded system applications with mobile and ubiquitous control access. It was a result of the integration of three different technologies: software component reuse, formal synthesis, and formal verification. Starting from user-specified UML models, automation was provided in model transformations, scheduling, verification, and code generation. VERTAF can be easily extended since new specification languages, scheduling algorithms, etc. can easily be integrated into it. Future extensions will include support for share-driven scheduling algorithms. More applications will also be developed using VERTAF. VERTAF will be enhanced in the future by considering more advanced features of real-time applications, such as: network delay, network protocols, and on-line task scheduling. Performance related features such as context switch time and rate, external events handling, I/O timing, mode changes, transient overloading, and setup time will also be incorporated into VERTAF in the future.

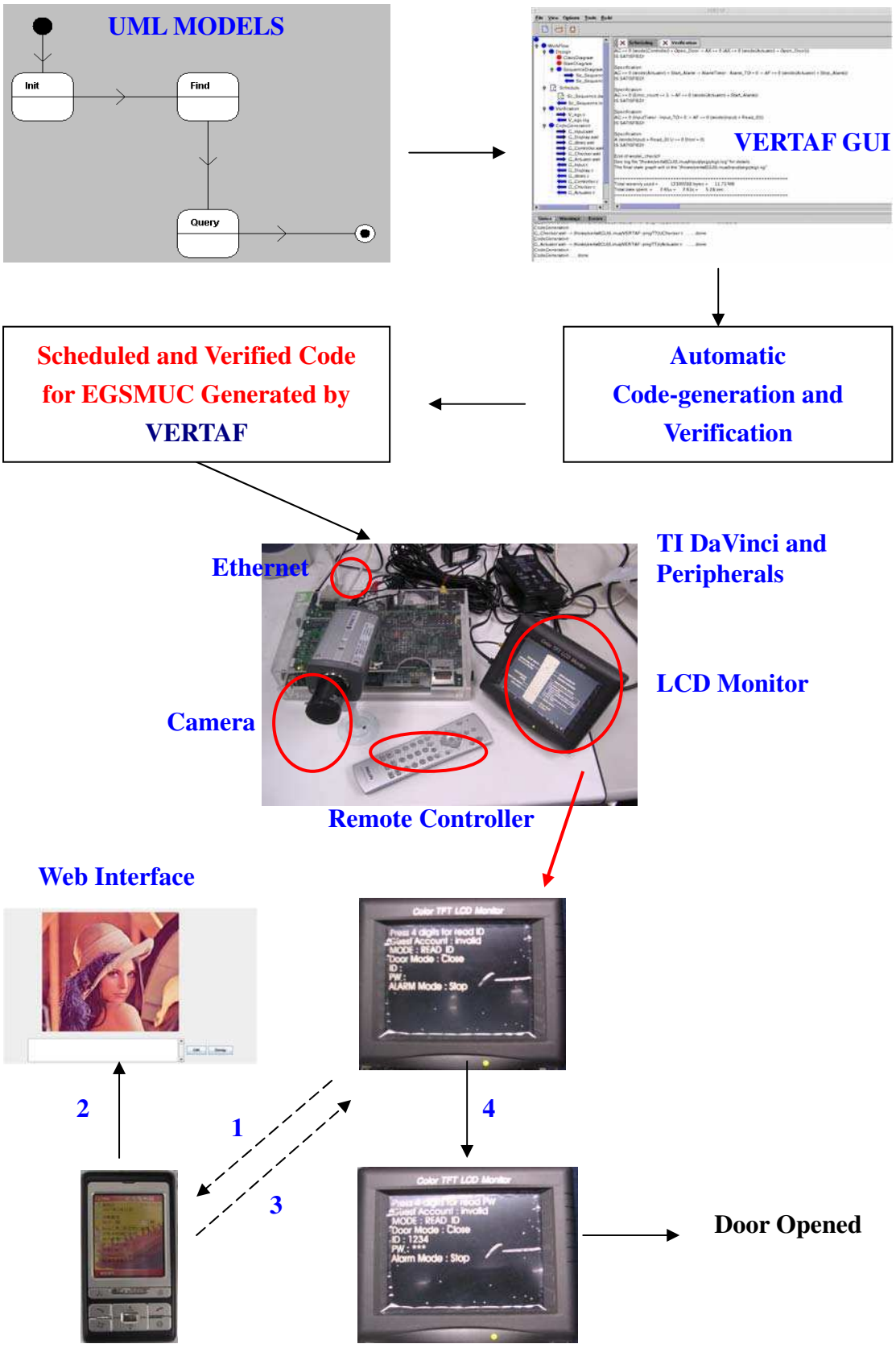

Figure 7. A Demo Flow of EGSMUC

References

[1] R. Alur and D. Dill. Automata for modeling real-time systems. *Theoretical Computer Science*, 126(2):183–236, April 1994.

[2] T. Amnell, E. Fersman, L. Mokrushin, P. Petterson, and W. Yi. TIMES: a tool for schedulability analysis and code generation of real-time systems. In *Proceedings of the 1st International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS)*, September 2003.

[3] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the Logics of Programs Workshop*, volume 131 of *LNCS*, pages 52–71. Springer Verlag, 1981.

[4] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[5] D. de Niz and R. Rajkumar. Time Weaver: A software-through-models framework for embedded real-time systems. In *Proceedings of the International Workshop on Languages, Compilers, and Tools for Embedded Systems*, pages 133–143, June 2003.

[6] B. Douglass. *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. Addison Wesley Longman, Inc., Reading, MA, USA, November 1999.

[7] T. Henzinger, S. Qadeer, and S. Rajamani. Decomposing refinement proofs using assume-guarantee reasoning. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'00)*, pages 245–252, 2000.

[8] P. Hsiung. RTFrame: An object-oriented application framework for real-time applications. In *Proceedings of the 27th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS'98)*, pages 138–147. IEEE Computer Society Press, September 1998.

[9] P. Hsiung. Embedded software verification in hardware-software codesign. *Journal of Systems Architecture - the Euromicro Journal*, 46(15):1435–1450, November 2000.

[10] P. Hsiung and S. Cheng. Automating formal modular verification of asynchronous real-time embedded systems. In *Proceedings of the 16th International Conference on VLSI Design, (VLSI'2003)*, pages 249–254. IEEE CS Press, January 2003.

[11] P. Hsiung and C. Lin. Synthesis of real-time embedded software with local and global deadlines. In *Proceedings of the 1st ACM/IEEE/IFIP International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS'2003)*, pages 114–119. ACM Press, October 2003.

[12] P. Hsiung, C. Lin, and T. Lee. Quasi-dynamic scheduling for the synthesis of real-time embedded software with local and global deadlines. In *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA'2003)*, February 2003.

[13] A. Knapp, S. Merz, and C. Rauh. Model checking timed UML state machines and collaboration. In *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 2469 of *LNCS*, pages 395–414. Springer Verlag, September 2002.

[14] S. Kodase, S. Wang, and K. Shin. Transforming structural model to runtime model of embedded software with real-time constraints. In *Proceedings of Design, Automation and Test in Europe Conference*, pages 170–175, March 2003.

[15] T. Kuan, W. See, and S. Chen. An object-oriented real-time framework and development environment. In *Proceedings OOPSLA'95 Workshop #18*, 1995.

[16] L. Lavazza. *A methodology for formalizing concepts underlying the DESS notation*. EUREKA-ITEA project (http://www.dess-itea.org), D 1.7.4, December 2001.

[17] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real time environment. *Journal of the Association for Computing Machinery*, 20:46–61, January 1973.

[18] F. M. and D. Schmidt. Object-oriented application frameworks. *Communications of the ACM, Special Issue on Object-Oriented Application Frameworks*, 40, October 1997.

[19] J. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the International Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer Verlag, 1982.

[20] J. Rumbaugh, G. Booch, and I. Jacobson. *The UML Reference Guide*. Addison Wesley Longman, 1999.

[21] M. Samek. *Practical Statecharts in C/C++ Quantum Programming for Embedded Systems*. CMP Books, 2002.

[22] D. Schmidt. Applying design patterns and frameworks to develop object-oriented communication software. In P. Salus, editor, *Handbook of Programming Languages*, volume I. MacMillan Computer Publishing, 1997.

[23] W. See and S. Chen. *Object-oriented real-time system framework*, chapter 16, pages 327–338. John Wiley, 2000.

[24] B. Selic, G. Gullekan, and P. Ward. *Real-time Object Oriented Modeling*. John Wiley and Sons, Inc., 1994.

[25] T. Shen. Assume-guarantee based formal verification of hierarchical software designs. Master's thesis, Dept. of CSIE, National Chung Cheng University, July 2003.

[26] F. Su and P. Hsiung. Extended quasi-static scheduling for formal synthesis and code generation of embedded software. In *Proceedings of the 10th IEEE/ACM International Symposium on Hardware/Software Codesign (CODES'02)*, pages 211–216. ACM Press, May 2002.

[27] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2002.

[28] F. Wang and P. Hsiung. Efficient and user-friendly verification. *IEEE Transactions on Computers*, 51(1):61–83, January 2002.

[29] S. Wang, S. Kodase, and K. Shin. Automating embedded software construction and analysis with design models. In *Proceedings of International Conference of Euro-uRapid*, December 2002.

[30] M. Zulkernine and R. Seviora. Assume-guarantee supervisor for concurrent systems. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, pages 1552–1560, April 2001.