Automatic Synthesis and Verification of Real-Time Embedded Software

Pao-Ann Hsiung and Shang-Wei Lin

Department of Computer Science and Information Engineering
National Chung-Cheng University, Chiayi, Taiwan, R.O.C.
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. In this work, we reveal the internal architecture and design flow of a newly proposed framework called *Verifiable Embedded Real-Time Application Framework* (VERTAF)[1], which integrates 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 design efforts as compared to that without VERTAF, which shows how high-level reuse of software components with automatic synthesis and verification increase design productivity.

Keywords: Application framework, code generation, real-time embedded software, scheduling, formal verification, software components, UML modeling

1 Introduction

With the proliferation of embedded 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 [4], [7]-[11], [13], [16]–[18], [20]. Nevertheless, the technology for a completely integrated design and verification environment is still relatively immature. This work demonstrates how the integration of software engineering

[1] This project was supported by a research project grant NSC92-2213-E-194-003 from the National Science Council, Taiwan

L.T. Yang et al. (Eds.): EUC 2004, LNCS 3207, pp. 12–21, 2004.

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.

Several issues are encountered in the development of an integrated design environment. First, we need to decide upon an architecture for the environment. 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 the environment as an object-oriented application framework [5], 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?
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.

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) [15] 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 [8].
3. System Models: For scheduling, we use variants of Petri Nets (PN) [10], [11] and for verification, we use Extended Timed Automata (ETA) [1], [11], 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 [10], [11] that generate program schedules for a single processor. For verification, we employ symbolic model checking [2], [3], [14] 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, we apply model-based, architecture-based, and function-based abstractions.

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.

The article is organized as follows. Section 2 describes the design and verification flow in VERTAF along with an illustration example. Section 3 presents the experimental results of an application example. Section 4 gives the final conclusions with some future work.

2 Design and Verification Flow in VERTAF

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 (EGS). EGS is an embedded system that controls the entrance to a building by identifying valid users through a voice recognition IC and control software that runs on a StrongARM 1100 microprocessor.

2.1 UML Modeling

After scrutiny of all diagrams in UML [4], [15], we have chosen three diagrams for a user to input as system specification models, namely class diagram, sequence

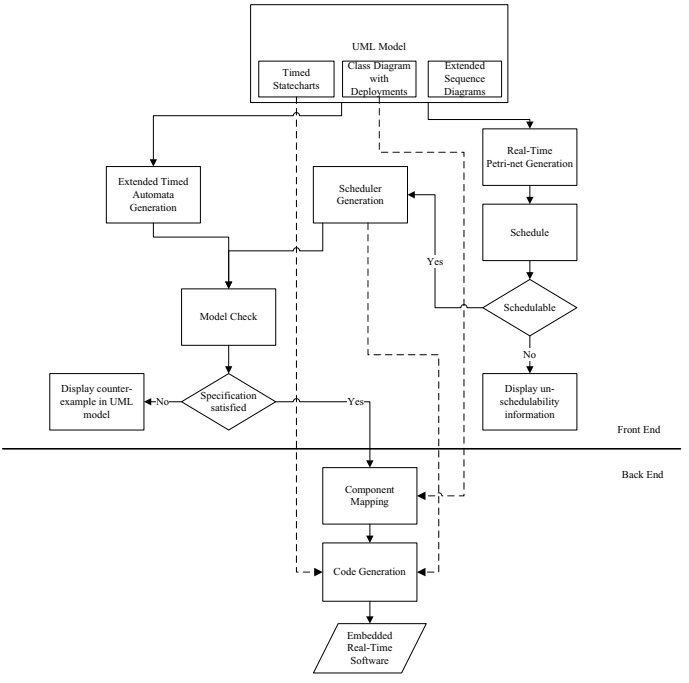

Fig. 1. Design and Verification Flow of VERTAF

diagram, and statechart. These diagrams were chosen such that information redundancy in user specifications is minimized and at the same time adequate expressiveness in user specifications is preserved. 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.

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 EGS example, some of the above diagrams are shown in Figures 2 and 3.

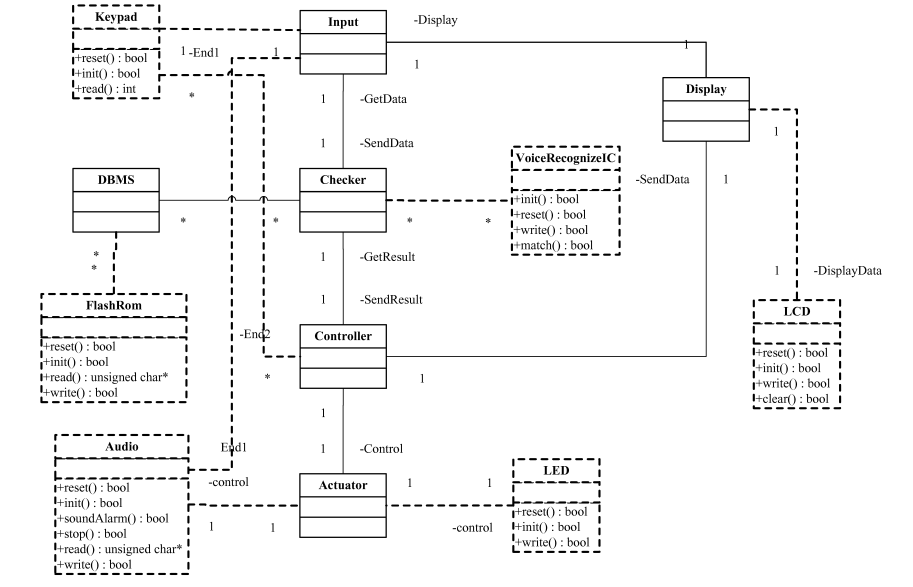

Fig. 2. Class Diagram with Deployment for Entrance Guard System

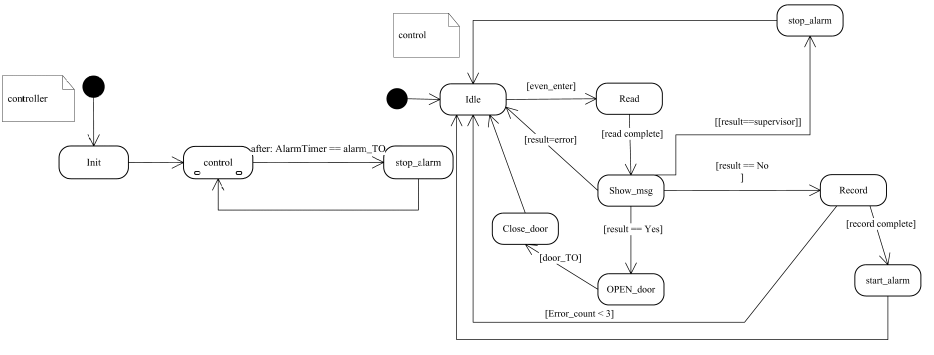

Fig. 3. Timed Statecharts for Controller in Entrance Guard System

UML is well-known for its informal and general-purpose semantics. Design guidelines are provided in VERTAF to a user such that the goal of correct-by-construction can be achieved as follows.

- Hardware deployments are desirable as they reflect the system architecture in which the generated real-time embedded software will execute and thus generated code will adhere to designer intent more precisely.
- If the behavior of an object cannot be represented by a simple statechart that has no more than four levels of hierarchy, then decompose the object.

- Overlapping behavior among scenarios often results in significant redundancy in sequence diagrams, hence either control structures may be used in a sequence diagram or a set of non-overlapping sequence diagrams may be inter-related with precedence constraints.
- Ensure the logical correctness of the relationships between class diagram and statecharts and between statecharts and sequence diagrams. The former relationship is represented by actions and events in statecharts that correspond to object methods in class diagram. The latter relationship is represented by state-markers in sequence diagrams that correspond to statechart states.

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) [10], [11] 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) [19] with real-time scheduling [12] 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, as follows.

1. A message in a sequence diagram is mapped to a set of Petri net nodes, including an incoming arc, a transition, an outgoing arc, and a place. If it is an initial message, no incoming arc is generated. If a message has a guard, the guard is associated to the incoming arc.
2. For each set of concurrent messages in a sequence diagram, a fork transition is first generated, which is then connected to a set of places that lead to a set of message mappings as described in Step (1) above.
3. If messages are sent in a loop, the Petri-nets corresponding to the messages in the loop are generated as described in Step (1) and connected in the given sequential order of the messages.
4. Different sequence diagrams are translated to different Petri-nets. If a Petri net has an ending transition which is the same as the initial transition of another Petri net, they are concatenated by merging the common transition.

5. Sequence diagrams that are inter-related by precedence constraints are first translated individually into independent Petri nets, which are then combined with a connecting place, that may act as a branch place when several sequence diagrams have a similar precedent.
6. An ending transition is appended to each Petri-net because otherwise there will be tokens that are never consumed resulting in infeasible scheduling.

For our running EGS 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 EGS 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) [20], 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 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 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.

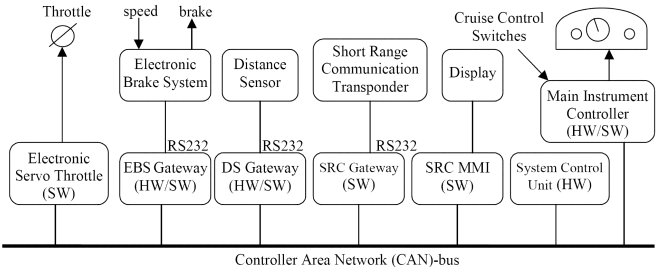

Fig. 4. AICC System Architecture

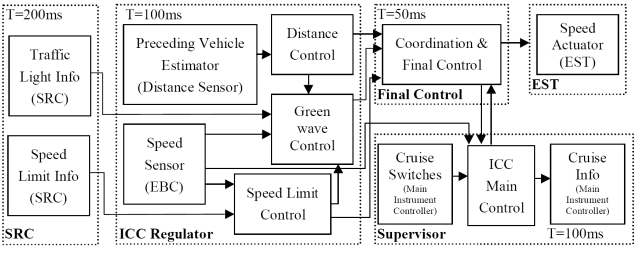

Fig. 5. AICC Call-Graph

Table 1. AICC Tasks

Index	Task Description	Object	p	e	d
1	Traffic Light Info	SRC	200	10	400
2	Speed Limit Info	SRC	200	10	400
3	Proceeding Vehicle Estimator	ICCReg	100	8	100
4	Speed Sensor	ICCReg	100	5	100
5	Distance Control	ICCReg	100	15	100
6	Green Wave Control	ICCReg	100	15	100
7	Speed Limit Control	ICCReg	100	15	100
8	Coordination & Final Control	Final_Control	50	20	50
9	Cruise Switches	Supervisor	100	15	100
10	ICC Main Control	Supervisor	100	20	100
11	Cruise Info	Supervisor	100	20	100
12	Speed Actuator	EST	50	5	50

SRC: Short Range Communication, ICCReg: ICC Regulator, EST: Electronic Servo Throttle, p: period, e: execution time, d: deadline

3 AICC Cruiser Application

An application developed with VERTAF is AICC (Autonomous Intelligent Cruise Controller) system application [6], which had been developed and in-

stalled in a Saab automobile by Hansson et al. The AICC system can receive information from road signs and adapt the speed of the vehicle to automatically follow speed limits. Also, with a vehicle in front cruising at lower speed the AICC adapts the speed and maintains safe distance. The AICC can also receive information from the roadside (e.g. from traffic lights) to calculate a speed profile which will reduce emission by avoiding stop and go at traffic lights. The system architecture consisting of both hardware (HW) and software (SW) is shown in Figure 4.

As shown in Figure 5, there are five domain objects specified by the designer of AICC for implementing a Basement system. As observed in Figure 5, each object may correspond (map) to one or more tasks. The tasks and the Call-Graph are as shown in Table 1 and Figure 5, respectively. There are totally 12 tasks performed by 5 application domain objects. There were 21 application framework objects specified by the designer. Totally, 26 objects were in the final program code generated. The average integration time per object was 0.5 day and the average learning time was amortized as 0.1 day for each designer using the framework. Without using the framework, the average integration time was 2 days for each object. This application took 5 days for 3 real-time system designers using VERTAF. The same application took the same designers 20 days to complete development a second time without VERTAF.

4 Conclusion

An object-oriented component-based application framework, called VERTAF, was proposed for real-time embedded systems application development. It was a result of the integration of 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. Future extensions will include support for share-driven scheduling algorithms, 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.

References

1. R. Alur and D. Dill, "Automata for modeling real-time systems," Theoretical Computer Science, Vol. 126, No. 2, pp. 183-236, April 1994.
2. E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in Proceedings of the Logics of Programs Workshop, LNCS Vol. 131, pp. 52-71, Springer Verlag, 1981.
3. E. M. Clarke, O. Grumberg, and D. A. Peled, Model Checking, MIT Press, 1999.
4. B. P. Douglass, Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns, Addison Wesley Longman, Inc., Reading, MA, USA, November 1999.

5. M. Fayad and D. Schmidt, "Object-oriented application frameworks," Communications of the ACM, Special Issue on Object-Oriented Application Frameworks, Vol. 40, October 1997.
6. H. A. Hansson, H. W. Lawson, M. Stromberg, and S. Larsson, "BASEMENT: A distributed real-time architecture for vehicle applications," Real-Time Systems, Vol. 11, No. 3, pp. 223-244, 1996.
7. P.-A. 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), pp. 138-147, IEEE Computer Society Press, September 1998.
8. P.-A. Hsiung, "Embedded software verification in hardware-software codesign," Journal of Systems Architecture - the Euromicro Journal, Vol. 46, No. 15, pp. 1435-1450, Elsevier Science, November 2000.
9. P.-A. Hsiung and S.-Y. Cheng, "Automating formal modular verification of asynchronous real-time embedded systems," in Proceedings of the 16th International Conference on VLSI Design, (VLSI'2003, New Delhi, India), pp. 249-254, IEEE CS Press, January 2003.
10. P.-A. Hsiung and C.-Y. 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, Newport Beach, CA, USA), pp. 114-119, ACM Press, October 2003.
11. P.-A. Hsiung, C.-Y. Lin, and T.-Y. 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, Tainan, Taiwan), February 2003.
12. C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real time environment," Journal of the Association for Computing Machinery, Vol. 20, pp. 46-61, January 1973.
13. 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, San-Diego, California, USA, pp. 133-143, June 2003.
14. J.-P. Queille and J. Sifakis, "Specification and verification of concurrent systems in CESAR," in Proceedings of the International Symposium on Programming, LNCS Vol. 137, pp. 337-351, Springer Verlag, 1982.
15. J. Rumbaugh, G. Booch, and I. Jacobson, The UML Reference Guide, Addison Wesley Longman, 1999.
16. M. Samek, Practical Statecharts in C/C++ Quantum Programming for Embedded Systems, CMP Books, 2002.
17. D. Schmidt, "Applying design patterns and frameworks to develop object-oriented communication software," Handbook of Programming Languages, Vol. I, 1997.
18. B. Selic, G. Gullekan, P. T. Ward, Real-time Object Oriented Modeling, John Wiley and Sons, Inc., 1994.
19. F.-S. Su and P.-A. 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, Colorado, USA), pp. 211-216, ACM Press, May 2002.
20. F. Wang and P.-A. Hsiung, "Efficient and user-friendly verification," IEEE Transactions on Computers, Vol. 51, No. 1, pp. 61-83, January 2002.