Hierarchical Integration of Runtime Models*

Cheng Xie[1], Wenzhi Chen[1], Jiaoying Shi[1], Lü Ye[2]

[1] College of Computer Science, Zhejiang University, Hangzhou 310027, P.R.China.
arthurxie@vip.sina.com,
wzchen@cad.zju.edu.cn, jyshi@cad.zju.edu.cn
[2] Department of Computer Science and Electronics Engineering, ZheJiang University of Science and Technology, Hangzhou 310012, P.R.China.
yelue2004@yahoo.com.cn

Abstract. The complexity of embedded applications is growing rapidly. Mainstream software technology is facing serious challenges for leaving out non-functional aspects of embedded systems. To achieve this goal, we have defined a component-based modeling and assembly infrastructure, Pcanel, that supports hierarchical integration of concurrent, runtime models. A key principal in Pcanel is its netlist, namely component connection network. Pcanel advocates netlist as global view of a systemic design, where the basic building block is component. The functionality of embedded system is modeled as netlist. The communication among components is modeled as token flow. The distribution of functionality on netlist is transparent from the runtime models, which makes communication refinement easier. When applied formal models to components, the resulting runtime netlist maintains assurance of diversified non-functional aspects, such as timing and deadlock. The infrastructure advances the synergy between design-time models and runtime models.

1 Introduction

The complexity of embedded applications is growing rapidly. Mainstream technology of embedded software development is facing serious challenges. While reliability standards for embedded software remain very high, many new requirements of embedded software are desirable, such as rapid deployment and update, much more dynamic reconfiguration, low-power mobile computing, multimedia signal processing, etc. Prevailing abstractions of computational systems leave out these non-functional aspects of embedded systems, so that the methods used for general purpose software require considerable adaptation for embedded software. To achieve this goal, we have defined a component-based modeling and assembly infrastructure, Pcanel, which supports hierarchical integration of concurrent runtime models for component-

* This work was supported in part by the Hi-Tech Research and Development Program of China (863 Program) under Component-based Embedded Operating System and Developing Environment (No.2004AA1Z2050), and Embedded Software Platform for Ethernet Switch (No. 2003AA1Z2160); In part by the Science and Technology Program of Zhejiang province under Novel Distributed and Real-time Embedded Software Platform (No. 2004C21059).

Z. Wu et al. (Eds.): ICESS 2004, LNCS 3605, pp. 589-594, 2005.

based embedded system construction. Pcanel emphasizes modeling, not any more interface definition, functional customization and hardware management.

Pcanel structures the development process into two hierarchies. The top hierarchy represents the abstract design of a system in terms of functionality, leaving out specific implementation details, such as hardware configuration, middleware and fault tolerance. The bottom hierarchy refines the design by realizing non-functional aspects in a structural and systematic way. For example, abstract components can be mapped to operating system processes, connections between components are supported by distributed middleware, and specific implementations are selected from component repository for requirements of fault-tolerance and real-time. The essential design problems are solved by modeling and analysis before final implementation. During recursive refinements viz. hierarchical integration of runtime models, the non-functional aspects including time, concurrency, correctness, reactivity, and heterogeneity are integrated into the design, until it can be finally synthesized to implementation. In this infrastructure, atomic component is built from computation blocks, and complex component is recursively built from composition of fine-grained components. Each component communicates with others under particular model of computation. The behavior of component is modeled as a set of transitions. A composite component is an encapsulated framework that consists of a runtime model and a graph of connected components. Through hierarchical integration of runtime models, Pcanel has great capability in separating systemic design from behavior specification, and capturing the requirements and constraints of the system.

2 Hierarchical Integration

Pcanel advocates netlist, namely component connection network, as global view of systemic design, where the basic building block is component. A component is a computational entity having a set of transitions. Components have interfaces defined by a collection of abstract input/output ports. Ports are shared states that allow components to communicate with each other via tokens. A set of connected ports represent a channel, through which a model drives the flow of tokens. A framework consists of a model and a graph of components, allowing for the observation and manipulation of the runtime states and behaviors internal of components.

Furthermore, to facilitate modularity, a framework itself, together with the components under its control, can be treated as a single component at a higher level of hierarchy, which means that the framework can be encapsulated to a composite component. Thus, a complete system configuration is a set of hierarchical compositions of models and components. Figure 1 shows two hierarchies of composition. The framework B is encapsulated into a composite component b by introducing more states. The transitions of {(3,4), (4,5), (5,6)} in B is abstracted as an atomic transition (1,2) in A. When applied formal models to components, the resulting composite component maintains assurance of diversified non-functional aspects, such as timing and deadlock.

Models are independent of implementation of components. Based on Pcanel, an embedded system can be built rapidly by reusing existed components and customizing netlist. Pcanel implements several models of computation for complex embedded

system design, including continuous time (CT), discrete event (DE), synchronous dataflow (SDF), communicating sequential processes (CSP), Priority-driven multitasking (PDM), and finite state machine (FSM), etc.

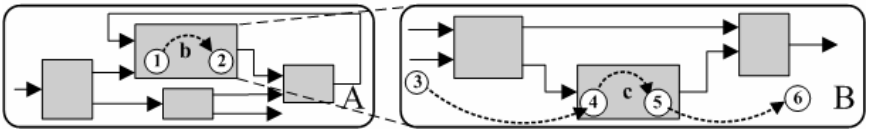

Fig. 1. Hierarchical Composition

The integration of hybrid runtime models of a two wheeled vehicle is shown in Figure 2. The vehicle, called Cyveh [11], is principally a self balancing machine with fully automated driving capabilities, whose wheels share a common axis. To implement the balance system of Cyveh, the control software is composed by hybrid runtime models including CT, DE, FSM, Modal and SDF.

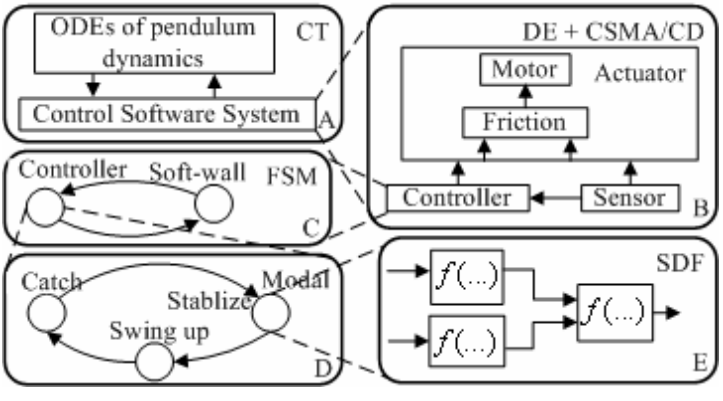

Fig. 2. Integration of runtime models in Cyveh

The top framework A is driven by CT model and contains two components, one is a set of differential equations modeling the physical pendulum dynamics, and the other is the control software. The framework B implements the control laws under DE model, which contains the actuator, the sensor and the controller. For heterogeneous road surface, a component serving friction compensation can be dynamically loaded into the actuator [4]. The framework C of the controller is driven by FSM model of two states. Any time the Cyveh enters a protected area, e.g. too close to other vehicles, the controller switches to the soft-wall state. A force is applied in the opposite direction to avoid collision. The framework D implements the normal operation within a modal model of three modes. Initially, the swing-up mode brings Cyveh from lean parking to upright position by energy control [5]. Once it is sufficiently close to upright position, the controller switches to the catch mode that slows down the pendulum body rotation before entering the third mode, stabilize. At last, the Cyveh keeps balance when the stabilize component is running. The framework E driven by SDF model implements a control algorithm that maintains the natural equilibrium point of the Cyveh system.

3 Component Netlist

Typical distributed embedded system consists of a network of nodes connected via bus or network. As the platform architecture shown in Figure 3, each node consists of the processor, an operating system, a dedicated communication layer, and one or more application transactions. The complete software can not independently from the hardware. The execution of software depends on the underlying processor architecture, memory mapping, bus of SoCs, or device registers. For reuse of components, hardware platform profile is included in the description of node.

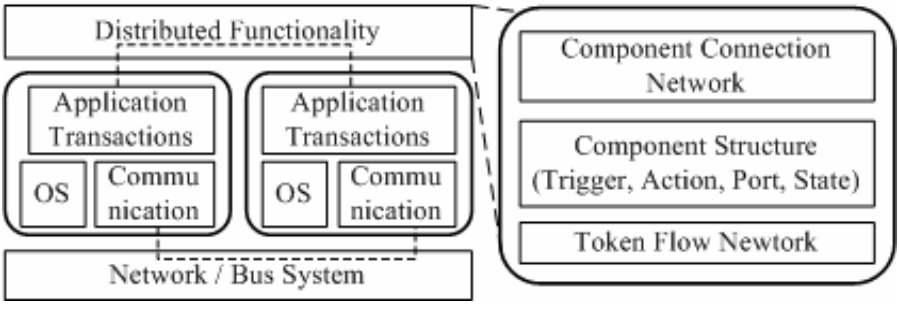

Fig. 3. Platform architecture

The behavior of a component is modeled as component structure defined in [11]. The behavior of a system is modeled as component netlist of hierarchical runtime models. The communication in netlist is modeled as token flow being carried out on token basis. The token flows are scheduled under models. In a hybrid system, hierarchical heterogeneous models cooperatively direct the token flows. A global token flow network can be constructed from the component netlist for analysis and verification of concurrent and real-time aspects in a hybrid system.

The distributed mapping from functionality to nodes is vertical to the netlist. The communication between components on spatially separated nodes is wrapped by the communication layer. The communication layer defines how software can be integrated with given components. The integrated system model may span hybrid bus systems, such as Controller Area Network and Local Interconnect Network. Since each reusable component is implemented with a set of transitions that uniquely define its functionality without side effects, components can be refined into the netlist based on their design specifications.

Models of computation are independent of implementation of components. Thus, Reusable components in integrated software are organized hierarchically to support integration with different models. A complete system configuration, i.e. the component connection network, is actually the synthesis result of hierarchical composition of reusable components. The netlist consists with the models of computation, thus allows for the observation and manipulation of the runtime states and behaviors internal of components. Such a netlist supports hierarchical composition, which is able to keep the global overview of the system.

4 Synergy Between Design and Runtime

Several approaches to the composition of software from components have been proposed in the literature [9, 7, 6]. An important contribution to this topic stems from the field of software architecture systems. Architecture systems introduce the notion of components, ports, and connectors as first class representations. In [8] a component model is used for embedded software in consumer electronic devices. In [1] a framework for dynamically reconfigurable real-time software is presented. It is based on the concept of so called Port Based Objects. However, most of the approaches proposed in the literature do not take into account the heterogeneous properties of software for hybrid systems.

Systematically integrating heterogeneous components is crucial to design large-scale distributed real-time systems. Many active research projects address this issue and influence our design. For example, [3] proposes a globally asynchronous and locally synchronous (GALS) architecture, that asynchronous message communication is used to maintain the synchronous semantics of execution of components and their composition. [10] integrates multi-rate time-triggered architecture with finite state machines. But most of these projects only integrate two models and assume a fixed containment relation between them. These architectures lack formal runtime models for composite components. Our infrastructure enables hierarchical heterogeneous compositions along well-defined models that are semantically separate from one another. In addition, unlike these approaches, our work has a strong emphasis on runtime systems.

It is important to advance the synergy between heterogeneous design environments and runtime systems. Design-time environments emphasize the understandability of models, syntax and semantics checking (like type systems), and component polymorphism. Ptolemy II [2] supports the modeling, simulation, and design of concurrent, real-time, embedded systems. It incorporates a number of models of computation (such as synchronous/reactive system, communicating sequential processes, finite state machine, continuous time, etc.) with semantics that allow domains to interoperate. On the other hand, runtime systems emphasize physical interface, performance, and footprint. Not all design-time models are suitable for direct implementation on runtime systems. Except for models that only are useful for modeling physical environment, certain models transformed to embedded software may be nondeterministic, inefficient and deadlock.

The rough approach of integrating heterogeneous runtime models is to implement them indirectly by a grand unified model. For example, it is possible to emulate most models on a time-synced priority-driven model provided in traditional RTOS. The methodology attempts to build a flat layer of abstraction fit for all applications. However, grand unified models are usually difficult of analysis and synthesis. In addition, an application usually does not need all the features provided by the grand unified model. Mixed features degrade performance and take overstaffed footprint.

Code generation approach adopted in Ptolemy II project is a migration path from certain design-time models to runtime models. The recent Ptolemy II release includes a limited prototype of code generation facility that will generate a stand-alone program of java class files for non-hierarchical SDF models. However, the code generation process is a very restricted solution for the wide heterogeneity and

irregularity of embedded systems, and the result program is not portable for variant operating systems.

We argue that a hierarchical runtime infrastructure natively supporting executable models will greatly help code generation and improve the quality of final software. A runtime system can utilize hardware support (such as SMP) and communication systems (such as CAN) to provide high responsible frameworks to applications. In addition, there are certain assumptions, like resource reservation and timing predictability, can only be achieved by OS-level runtime systems, but not easily by stand-along programs.

5 Conclusion

Noticing a wide variety of design-time models for distributed, real-time, embedded systems, this paper motivates a component-based modeling and assembly infrastructure, Pcanel, to integrate heterogeneous executable models and support composition of components. Pcanel proposes a runtime infrastructure for constructing responsible systems through runtime models integration. A key principal in the infrastructure is its component netlist, which makes the runtime system responsible for the distributed functionality. The novel design of Pcanel advances the synergy between heterogeneous design environments and runtime systems.

References

1. D.B.Stewart, R.A.Volpe, and P.K.Khosla. Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects. IEEE Transactions on Software Engineering, 1997.
2. E. A. Lee. Overview of the Ptolemy Project. Technical Memorandum UCB/ERL M03/25, University of California, Berkeley, Berkeley, July 2, 2003.
3. E. Cheong, J. Liebman, J. Liu, and F. Zhao. TinyGALS: A programming model for event-driven embedded systems. Proceedings of the Eighteenth Annual ACM Symposium on Applied Computing, 2003.
4. E. Garcia, P. Gonzalez, and C Canudas de Wit. Velocity dependence in the cyclic friction arising with gears. International Journal of Robotics Research, 21(9):761–771, 2002.
5. K. Astrom and K. Furuta. Swinging up a pendulum by energy control. IFAC 13th World Congress, San Francisco, California, 1996.
6. M. Shaw and D. Garlan. Software Architecture - Perspectives on an Emerging Discipline. Prentice Hall, 1996.
7. Paul C. Clements. A survey of architecture description languages. International Workshop on Software Specification and Design, 1996.
8. Rob van Ommering, Frank van der Linden, Jeff Kramer, and Jeff Magee. The koala component model for consumer electronics software. IEEE Computer, 2000.
9. Robert Allen and David Garlan. A formal basis for architectural connection. ACM Transactions on Software Engineering and Methodology, 6(3):213– 49, July 1997.
10. Thomas A. Henzinger, Christoph M. Kirsch, Marco A. Sanvido, and Wolfgang Pree. From control models to real-time code using Giotto. IEEE Control Systems Magazine, 2003.
11. Cheng Xie, Wenzhi Chen, Jiaoying Shi. Pcanel: A Model Driven Component Framework. IEEE Conference on Systems, Man and Cybernetics, 2004.