A FORMAL ARCHITECTURAL SPECIFICATION MODEL FOR REAL-TIME SYSTEMS DESIGN[1]

JIACUN WANG and YI DENG

School of Computer Science
Florida International University
Miami, FL33165
e-mail:{wangji, deng}@cs.fiu.edu

XUDONG HE

Department of Computer Science
North Dakota State University
Fargo, ND58105
email: hex@grant.cs.ndsu.nodak.edu

ABSTRACT

A Real-time Architectural Specification (RAS) model is presented. The objective of RAS is to establish a formal foundation that will enable us to integrate existing rich but fragmented formal techniques for system specification and verification into practical and scaleable formal engineering methods to support the design and development of highly reliable real-time distributed systems. The contribution of RAS is twofold: First, it offers an incremental and more scaleable approach for design modeling. Second, it provides a systematic way to enforce and maintain the consistency of the real-time system constraints in every step of the design process. These two features together make RAS a suitable model for the design of real-time systems.

Keywords: Software architecture, formal methods, real-time systems, time Petri nets, real-time computational tree logic.

1. INTRODUCTION

Real-time systems must not only behave functionally correct, but also satisfy real-time constraints. Due to the complexity and stringent time dependent requirements, these systems are extremely difficult and costly to develop.

To ensure the correctness of real-time systems, many formal methods have been developed over the past two decades, which help remove ambiguity and inconsistency in the real-time system development process [4]. However, their wide acceptance as a practical and effective software development approach still remains to be seen. Among many reasons, formal models are difficult to create, change and understand. The majority of published results on formal methods only used very small toy problems to demonstrate the intrinsic features of the formal models and the associated approaches and techniques, but failed in showing how to scale the techniques to realistic sized problems. Comparatively speaking, there have been far more research efforts and papers addressing theoretical issues than how to incorporate theories into software engineering methods. There is a lack of systematic ways to show how these techniques can work together in complex projects. Although many of the above problems are well-documented in [4], research efforts towards representing and formalizing system development processes have only emerged in the recent years.

The paper aims to address some of the above problems with a formal systematic approach (a Real-time Architectural Specification model – RAS in the sequel) to support real-time system development. RAS is built on complementary dual formalisms – time Petri nets [2] (TPN in the sequel) for specifying system behavior and real-time computation tree logic [6] (RTCTL in the sequel) for defining system constraints in a way that facilitates the compositional analysis of system properties. RAS supports a hierarchical component-based real-time system development approach that provides system scalability and maintainability, and ensures the consistency of system constraints among related components at the same abstraction level as well as different abstraction levels. By integrating TPN and RTCTL into a coherent architectural model, RAS provides a formal approach to systematically maintain time-critical system constraints during architecture refinements. Therefore, RAS not only facilitates the design traceability (allow us to determine the acceptable behavior of a system component with regard to the time constraint at any design level), but also supports a flexible and evolutionary design process such that system components can be refined and analyzed incrementally based on the constraints imposed on them.

The advantages of using complementary dual formal methods have been known for a long time. Many research

1. This work was supported in part by the NSF under Grant No. HDR-9707076, by Air Force Office of Scientific Research under Grant No. F49620-96-1-0221, and by Army Research Office under grant No. DAAG55-98-1-0428. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official polices or endorsements either expressed or implied by the above named Agencies.

efforts exploring approaches in integrating complementary dual methods have been published, especially those integrating Petri nets and temporal logic [7, 9]. However, most of the approaches do not support hierarchical component-based development. On the other hand, there are research efforts specifically dealing with hierarchical and compositional formal specifications, such as [1, 3, 8, 10]. However, none of the above approaches provides the refinement and/or composition mechanisms as general as our approach.

2. CONCEPTUAL FRAMEWORK OF RAS MODEL

Our goal is to develop both a rigorous approach to enhance the integrity of design and an evolutionary process to control complexity in system modeling and analysis. To the end, RAS models a distributed system as a multi-leveled composition of components and real-time constraints that the components and their compositions must satisfy at every design level. As refinement to the system design goes on, these architectural constraints are also decomposed and propagated to the lower levels of system architecture (possibly new constraints are introduced). The consistency between higher-level and lower-level design constraints are either automatically maintained or verified to ensure consistency and progressive characteristics of analysis.

More specifically, an RAS model consists of three basic elements: *component models*, *inter-component connections* (*connections* in brief), and *architectural constraints* (*constraints* in brief) organized into multi-design levels. The component models describe the real-time behavior and communication interface of the components. The connections specify how the components interact with each other and, in turn, form the *system composition model*. Finally, the constraints define real-time system requirements imposed on the components and connections. All connections are defined using only communication interfaces, which gives us the flexibility to change the design of individual components without voiding the analysis of the entire system.

A typical RAS model is illustrated in Figure 1. The high-level design has three components – *A*1, *A*2, and *A*3. Component *A*3 is further refined at the next design level into the composition of components *B*1, *B*2, and *B*3. The design at any level must satisfy the constraints specified at that level. A component model has two parts: (1) *communication ports* (denoted graphically by half circles), including *input ports* (e.g., *port*7) and *output ports* (e.g., *port*8), and (2) a TPN that describes the time-dependent, operational behavior of the component, that is, it defines the semantics associated with the ports. The communication between a component and its environment is solely through the ports. A connection represents a channel of interaction between components. It is modeled by a simple TPN and defines the direction of message flow and delay in the channel. For example, components *A*1 and *A*2 have a request-reply relationship that is modeled by the bi-directional channel. At any given design level, ports can also be divided into *external ports*, e.g. *port*1, and *internal ports*, e.g. *port*3 and *port*4. An internal port describes interaction between components within a system (or subsystem), while an external port describes the inputs and outputs from and to the

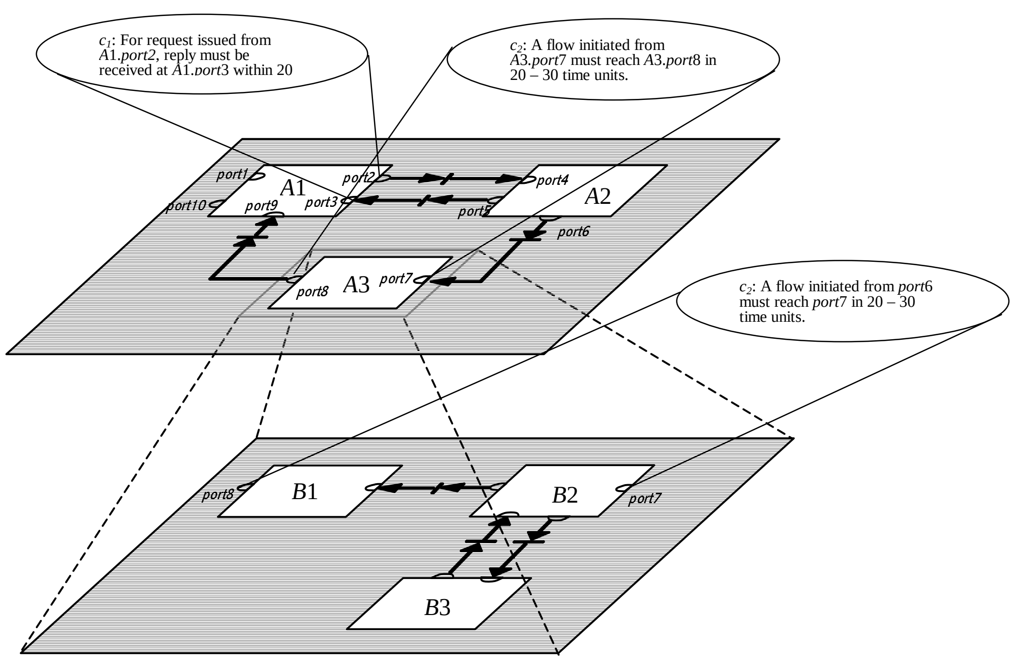

Figure 1. Framework of the RAS model.

environment of a system (or subsystem).

In addition to serving as a component's communication interface, the ports also provide the *linkage* between the operational design (components and connections) and the descriptive architectural constraints. In particular, time-critical system constraints are specified by RTCTL formulas defined over ports, where each port represents an *atomic proposition*, which is true at the moment that a token arrives in the port. These ports constitute the alphabet (atomic propositions) of the RTCTL formulas that define the constraints. All constraints are specified using ports only, no internal information about the components is revealed. For example, constraint c_1 limits the time taken in a request-reply interaction between $A1$ and with $A2$, and constraint c_2 specifies the required processing time of component $A3$. This arrangement is critical to achieve the goals of incremental modeling and analysis described earlier, as it ensures that the component designs can be treated as black-boxes in the construction, understanding and analysis of a system's architecture. As more detailed system architecture is constructed by decomposing one of the system's components, it becomes possible for us guarantee the satisfaction of system-wide requirements by verifying the new sub-architecture meets its obligation imposed on its interface.

From the viewpoint of origination, architectural constraints are divided into two groups, one is *original constraints*, which reflect the user's requirements on the system under developed, and the other is *derived constraints*, which are intermediate constraints derived from original constraints — Notice that while original constraints may indeed be few in number, maintaining functionally correct original values may involve a large set of interacting components. Thus, to ensure these original constraints are satisfied, each of these components will, in turn, be subject to their own derived constraints. From the viewpoint of function, architectural constraints in an RAS model can also be logically divided into three classes: *component constraints*, *environmental constraints*, and *path constraints*. A component constraint describes a time-critical property of a component that its environment expects from it. An environmental constraint describes a time-critical property that a component expects from its environment. A path constraint describes a time-critical property for message transmission across components, which normally describes a system-wide (or subsystem-wide) timing requirement.

In order to be able to design a component independent of the rest of a system, it is desirable that there is a set of constraints which completely describe what properties the component are expected by its environment, and what properties the component expects from its environment. Such a property is extremely expected when we conduct designs of complex real-time systems.

3. A METHOD FOR ARCHITECTURAL MODELING, REFINEMENT AND ANALYSIS

The RAS representation does not dictate a specific method of system modeling, refinement and design. However, a well-defined modeling and analysis method that is easy to follow and that can be integrated with software development process is essential to gain the application of formal techniques in practical system development. Such a modeling method is an integral part of a formal method of system design and formal methods research. In this section, we describe a concrete procedure that can be used for build and analyze RAS architectural models, which provides both a systematic way to enhance the integrity of design and an evolutionary process to control complexity in system modeling and analysis. It consists of the following major steps.

Step 1. Design RAS top-level architecture of a system.

In this step, top-level system architecture is formed by decomposing the system into several subsystems or components, as well as the connections between components. For each component, its input ports and output ports are defined.

Step 2. Specify architectural constraints.

In this step, we first use RTCTL formulae to formally represent the system-wide original requirements. As stated in Section 2, each requirement is specified using ports only. System-wide time-critical requirements are typically captured by path-constraints. The importance of formalizing these original requirements is twofold: First, the global system requirements are transformed into specific constraints on this particular architecture, more precisely, constraints imposed on the subsystems and connections between the subsystems. Second, by formalizing time-critical requirements in terms of architectural constraints (based solely on the component interfaces), it not only removes ambiguity in the description, but also makes it easier to detect possible inconsistency or conflict between different (competing) requirements.

Step 3. Design components.

In this step, we use TPN as a tool to define the operational behavior of each component. Using the Petri net design methodologies, we can synthesize an ordinary Petri net model for a component based on its operations relationship among these operations. After we get such an ordinary Petri net, giving time requirements for various operation results in a TPN model, in which every transition is associated with an appropriate time delay interval.

Upon the completion of this step, we now have a complete, executable and analyzable formal model that represents the top-level design of the system.

Step 4. Verify architectural constraints.

In this step, we check if the operational model of the system is consistent with its constraints.

The modular nature of RAS model and its emphasis on maintaining a strong correlation between design and requirements provide a natural support for incremental verification and for enforcing conformance of the design to the requirements. We have developed a two level incremental verification technique that helps achieve these benefits [5, 11], the first level being *component-oriented*, while the second level *system-oriented*. The process of verification is driven by showing that the components and their composition satisfy their corresponding constraints at every design level.

The basic idea of component-oriented verification is to check each component against the constraints defined only on this component provided that for each component there is a set of constraints completely describe what properties the component are expected by its environment, and what properties the component expects from its environment. If all these constraints are satisfied, then the component is satisfied; If all components are satisfied, then the system are satisfied.

By system-oriented verification we check the system design against both original constraints and derived constraints, but only the violation of original constraints terminates the verification process. After the completion of the verification, if all original constraints are satisfied while some derived constraints are violated, we readjust these derived constraints such that they are satisfied under the current design, and then, we can safely propagate these constraints to the lower level design. To this end, we developed a set of component-level reduction rules for TPN, which can be found in [11].

Step 5. Incrementally model the system.

In this step, each component is checked against its specification. If a component is too abstract for direct implementation, we use the corresponding operational model of the component as the basis and repeat step 1-5 to develop a more detailed architectural design for the component. If necessary, more than one sub-architecture that conform to horizontal interface consistency conditions can be developed and plugged in the position of the component model to evaluate different design alternatives.

4. CONCLUSION

This work is a step toward an integrated, practical and scaleable formal engineering method for real-time distributed systems design. The contribution of RAS is the approach for architectural modeling and design that it represents. The essence of this approach is integration – the integration of architectural design and refinement with the propagation of time-critical system constraints into a coherent modeling and analysis framework, and the integration existing rich, but fragmented, formal specification and verification techniques that implements the framework.

REFERENCE

[1] R. Bastide, C. Blanc and P. Palanque, Cooperative objects: A concurrent Petri-net based, objected-oriented language, *Proc. of the IEEE International Conference on System, Man and Cybernetics*, 1993, 286-291.

[2] B. Berthomieu, and M. Diaz, Modeling abd verification of time dependent systems using time Petri nets, *IEEE Transactions on Software Engineering*, vol.17, no.3, 1991, 259-273.

[3] G. Bucci and E. Member, Compositional validation of time-critical systems using communicating time Petri nets, *IEEE Transactions on Software Engineering*, vol.21, no.12, 1995, 969-992.

[4] D. Graigen, S. Gerhart, and T. Ralston, Formal methods reality check: industrial usage, *IEEE Transactions on Software Engineering*, vol.21, no.2, 1995, 90-98.

[5] Y. Deng, J. Wang, and R. Sinha, Incremental Architectural Modeling and Verification of Real Time Concurrent systems, *Proc. of the 2nd IEEE International Conference on Formal Engineering Methods*, Brisbane, Australia, December 1998.

[6] E.A. Emerson, A,K, Mok, A,P. Sistla, and J. Srinivasian, Quantitative temporal reasoning, *Real Time Systems*, vol.4, 1992, 331-352.

[7] M. Felder, D. Mandrioli, and A. Morzenti, Proving properties of real-time systems through logical specifications and Petri net models, *IEEE Transactions on Software Engineering*, vol.20, no.2, 1994, 127-141.

[8] X. He and Y. Ding, Object orientation in hierarchical predicate transition nets, *Advances in Petri Nets, Lecture Notes in Computer Science* (ed. G. Ahga), 1998, in press.

[9] X. He and J.A.N. Lee, Integrating predicate transition nets with first order temporal logic in the specification and verification of concurrent systems, *Formal Aspects of Computing*, vol.2, no.3, 1990, 226-246.

[10] Y.K. Lee and S.J. Park, OPNets: an object-oriented high-level Petri net model for real-timesystem modeling, *Journal of Systems and Software*, vol.20, 1993, 69-86.

[11] J. Wang and Y. Deng, Incremental modeling and verification of flexible manufacturing systems, *Journal of Intelligent Manufacturing*, 1998, to appear.