Integrated Architectural Modeling and Analysis for High-Assurance Command and Control System Design[1]

Yi Deng and Jiacun Wang
School of Computer Science
Florida International University
Miami, FL 33199
e-mail: {deng, wangji}@cs.fiu.edu

Abstract

A Real-Time Architectural Specification (RAS) approach and its application to command and control (C2) systems are presented. The objective 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 provides a formal system that integrates system's timing requirements and the propagation of the requirements into the process of architectural modeling and design and provides a systematic way to enforce that the requirements are met in every step of the design process. Second, it offers an incremental and more scaleable approach for design modeling. These two features together make RAS a suitable model for the design of C2 systems. Based on these features, we further present an incremental method for verifying timing properties of a RAS model that helps to reduce the complexity of analysis both at a given design level or across different design levels.

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

1. Introduction

A command and control (C2) system is a complex integration of distributed hardware and software components. A C2 system is a typical real-time system. The system reaction time, one of the most important performance measures of any C2 system, closely relates to the time delays associated with the execution of the generic C2 functions, such as detection, tracking, discrimination, etc. Delays in execution of any of these functions may degrade the system effectiveness, result in inefficient use of battle space, cause too many weapons to be assigned against the same target, and perhaps allow many targets to leak through a particular defense zone.

There are a number of tools and techniques that were widely used to support the modeling and design of C2 systems. They range from the use of informal block and function-flow diagrams and building simulation models [1, 15] to the use of formal specification and verification methods such as Petri nets [2, 12, 14, 20], statecharts [10] and real-time and temporal logics [8]. Block and function-flow diagrams are used to indicate interconnection of physical devices. They are simple and easy to use, but they are inadequate to precisely capture information flow, the sequence of events, the essential precedence relationships, and the time delays that are so crucial for C2 systems. Large-scale microscopic simulations are time-consuming, expensive, and not well suited for the analysis, design, and evaluation of alternate distributed C2 architectures. In addition, model validation is difficult by simulation alone. Because of their rigorous nature, the formal techniques have, in particular, been considered a promising way to raise the assurance in design. However, despite intensive study and significant advances in theoretical fronts, widely spread application of these formal techniques in practical system development still remains to be seen. A major cause of this problem is that the investigations for formal techniques are largely fragmented and there is lack of guidance as how to integrate these techniques as an inherent part of the development process.

In this paper, we explore a new approach to integrate theoretically rich but fragmented formal techniques into practical and scalable modeling and design methods for real-time distributed C2 systems. In particular, the focus is on how to systematically ensure time-critical system

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.

requirements are met in architectural design and yet to reduce the complexity of formal modeling and analysis. Our approach is based on two basic hypotheses: (1) The ability to systematically maintain a strong correlation between time-critical system requirements and the design of the system at every step of the design process is the basis to achieve high-assurance in design and to make systems changeable and extensible. An effective way to achieve such strong design traceability is to incorporate these time-critical requirements (or constraints) as an integral part of design modeling. (2) Incremental analysis sensibly integrated as a part of design process is a powerful means to control complexity and cost in analysis.

Based on these principles, we have developed the Real-time Architectural Specification (RAS) model [7, 19]. RAS is built on top of Time Petri Nets (TPN) [4] and Real-Time Computational Tree Logic (RTCTL) [8]. Petri nets are a well-known operational model best suited for modeling the control of distributed systems but cumbersome for specifying rules and constraints. By contrast, temporal logic, a popular descriptive formalism, is best suited for describing rules and constraints but not control and composition of systems. By integrating them into a coherent architectural model, RAS provides a formal system to systematically link the propagation of time-critical system requirements with the refinement of architectural design. In RAS, the propagation of these requirements goes in lockstep with the refinement of system design. In the process, these system-wide timing requirements are decomposed into timing constraints imposed on system components and the connections between the components at different design levels. These system-wide timing requirements are met by showing incrementally that the timing constraints imposed on the system components and their connections are satisfied by the design. Doing so, we not only guarantee design traceability, that is, the ability to determine at any design level the role a system component plays in satisfying a particular system requirement, but also support a flexible and evolutionary process of design in which system components can be refined, analyzed and/or verified incrementally based on the constraint imposed on them. Throughout the paper, these features will be demonstrated through a case study of architectural modeling and verification of a tactic anti-air C2 system.

The advantages of using complementary dual formal methods have been known for a long time. Many research efforts exploring approaches in integrating complementary dual methods have been published, especially those integrating Petri nets and temporal logic [9, 11]. 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 [4, 5]. However, none of them provides the refinement and/or composition mechanisms as general as our approach.

The rest of the paper is organized as follows: An overview of RAS model, including an example and formal definition, is given in Section 2. In Section 3, we illustrate the use of RAS to give an incremental modeling of a tactic anti-air C2 system. In particular, in Section 3.5, we briefly describe a constraint-based incremental timing verification technique based on the RAS model, which helps us to reduce the complexity of verification both at a given design level and across different design levels. Finally, we conclude the paper in Section 4.

2. Overview of RAS Approach

Our goal is to develop an approach which systematically maintains a strong correlation between time-critical system requirements and the design of the system at every step of the design process so as to achieve high-assurance in design and to make systems changeable and extensible. To the end, RAS specifies a distributed system with a set of component-based models, each of which describes the system at a given design level. Horizontally at any given design level, the model captures not only the operational property (components and connections) but also the descriptive architectural constraints. Vertically, there is a mapping between any two consecutive design levels, which provides the traceability across different design levels. To enforce a user to systematically maintain a strong correlation between time-critical system requirements and the design of the system at every step of the design process, the design process is guided by the following rules from the beginning to the end:

(1) At any design level, all constraints are consistent, that is, the true of any constraints will not necessary lead to the false of any other constraints.

(2) Before the design at any level is finished, make sure the operational behavior model is consistent with all architectural constraints imposed on it by using verification technique.

(3) Suppose that a component in design level k corresponds to sub-architecture in design level k+1, then the sub-architecture must take all ports of the component as all its external ports, as will be illustrated in Figure 1.

(4) Suppose that a component in design level k corresponds to sub-architecture in design level k+1, then sub-architecture must conform to all constraints which the component are subject to, as will be illustrated in Figure 1.

2.1 Conceptual Framework of RAS Model

The cardinal part of RAS approach is RAS models. 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 environment of a system (or subsystem).

2.2 Relationship between Design and Requirements in RAS Approach

In an RAS model, 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 *A*1 and with *A*2, and constraint c_2 specifies the required processing time of component

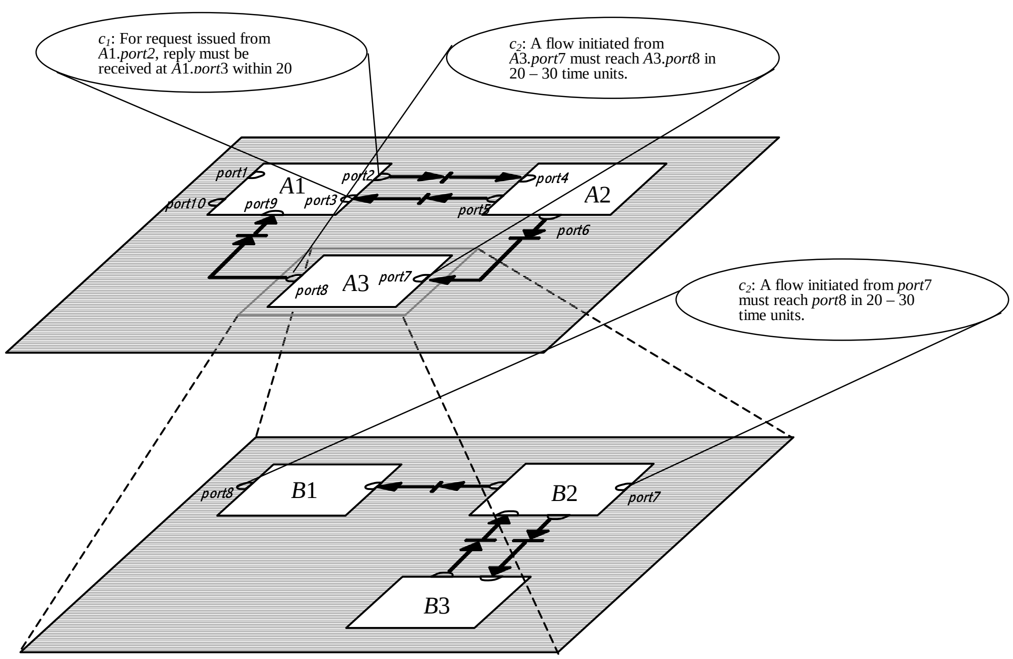

Figure 1. Framework of the RAS model.

*A*3. 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.

More important, the strong linkage is maintained during the whole process of system design. Suppose that we want to refine a component, say A3, into a sub-architecture composed of component B1, B2 and B3, as indicated in the Figure 1. Then, all the ports of A3, i.e., *port7* and *port8*, are inherited in the sub-architecture, and as will be discussed in Section 3, constraints related to the component, e.g., constraint C2, are either decomposed or inherited in the sub-architecture, which helps to ensure the consistency of constraints across design levels. Therefore, as more detailed system architecture is constructed by decomposing one of the system's components, it becomes possible for us to guarantee the satisfaction of system-wide requirements by verifying the new sub-architecture meets its obligation imposed on its interface.

2.3 Classification of Architectural Constraints

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.

The functional classification brings the following benefits: 1) Under the condition that all constraints are verified, if we want to change the design of an individual component, we only need to re-verify its component constraints and limited number of related constraints. 2) It separates the requirements on components from constraints on connections between them. Consequently, they can be analyzed separately. 3) Path constraints provide a natural way to specify system-wide properties. 4) Environmental constraints explicitly define what behavior a component (or the system) can expect from its environment, and thus eliminates the need to consider those disallowed events and states in design and analysis. They are different from the rest of the constraints in the sense that verification of a design entails showing that the other constraints are satisfied whenever the environmental constraints are satisfied. This effectively reduces the state space of a system architecture, which, in turn, reduces the complexity of design and analysis.

2.4 A Method for Architectural Design, 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.

(1) **Design RAS interface 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.

(2) **Specify architectural constraints.** We use RTCTL formulae to formally represent the system's constraints. Each constraint is specified using ports only. In this step, system-wide time-critical requirements are decomposed into architectural constraints associated with newly created components and their connections.

(3) **Construct operational model for each component.** In this step, we use TPN to formally represent the operational behavior of each component.

(4) **Verify architectural constraints**. In this step, we check if the operational model of the system is consistent with its constraints. If some constraints are found violated, we need to (partly) redesign the system, or adjust some constraints if allowable. We will address this issue in Section 3.5.

(5) **Incrementally model the system**. If a component is too abstract for direct implementation, we use the corresponding operational model of the component as

the basis and repeat step 1-4 to develop a more detailed architectural design for the component.

3. Incremental Constraint-Driven Modeling and Analysis of C2 System

In this section, we show how to use the RAS representation and the modeling method described above to model and analyze a tactic anti-air C2 system, and discuss the benefits of this modeling approach in achieve the goals stated at the beginning of the paper.

The C2 process can be decomposed into a set of generally accepted C2 functions that must be executed (sometimes serially and sometimes in parallel and, in general, in an asynchronous manner) to ensure mission success. The functions related to a defensive battle management C2 system include [2]: *threat detection*, *discrimination*, *identification and tracking*, *threatening assessment*, *battle planning*, *weapon-to-target assignment* (or *fire assignment*), *engagement control*, and *damage assessment*.

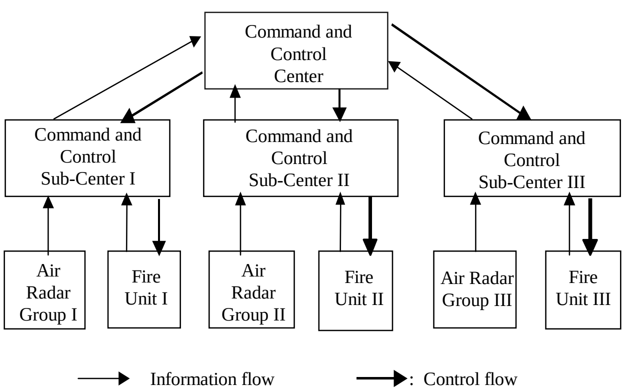

Figure 2. A generic tactic anti-air C2 systems with two level command and control centers.

3.1 Description of the Tactic Anti-Air C2 System

A typical structure of a tactic anti-air C2 system with two levels of command and control centers is shown in Figure 2, which consists of one first-level command center (indicated as *C2 center*) and three second-level command centers (indicated as *sub-centers*). A pair of (C2 center, sub-center) may be (division, regiment) or (brigade, battalion). They are geographically dispersed due to environmental and survivability reasons, which contributes to the distributed architecture of C2 organization.

We have pointed out in Section 1 that the performance of a C2 system is like a race against time between the moving physical entities and the information variables. In this case study, we will focus on the requirements of the time delays in the execution of the system functions, and how to construct an architectural model that meets these delay requirements. These requirements include:

(R1) The system reaction time, i.e., the time delay from an enemy intelligence message being received by any subcenter to a fire command agaisnt the enemy being issued by a corresponding fire unit, must be less or equal to 45 time units.

(R2) Since a C2 system is a closed-loop system, a constraint reflecting the time limitation for the feedback of damage assessment results should be included. This is captured by the requirement that the time delay from a detail fire assignment scheme being made out by a sub-center to the result of the damage assessment referred to the execution of this scheme being received by the same sub-center must be less or equal to 20 time units.

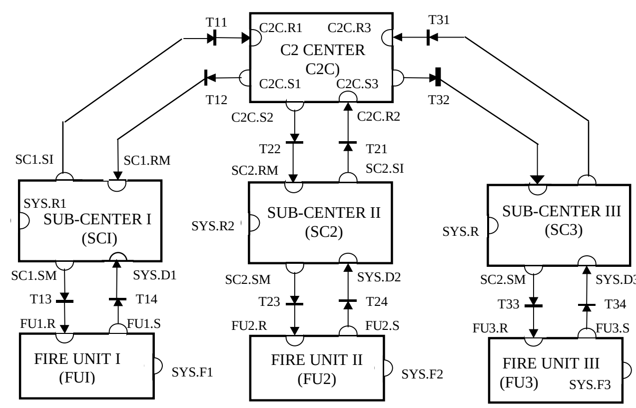

Figure 3 RAS interface of the tactic anti-air C2 system.

(R3) Since the bottleneck for information processing is often located in component C2 Center, the component is always asked to respond as soon as possible. This is captured by the requirement that the whole processing time for a group of messages from the three sub-centers must be less or equal to 22 time units.

All these requirements are applied under the assumption that no waiting time exists for any message at any processing stage. In order to simplify analysis, we also make the following assumptions for the system:

(A1) All the three sub-centers have identical topology and timing properties, and all the three fire units have identical topology and timing properties.

(A2) All the three sub-centers access intelligence messages received from their corresponding radar groups periodically at the same time points.

3.2 RAS Architecture of the C2 System

In order to capture the profile of the RAS model of the C2 system, we first define the interaction interface for the system's components, which is shown as in Figure 3. There are total seven components in the system: a C2 Center (*C2C*), three Sub-Centers (*SC1*, *SC2*, *SC3*) and three Fire Units (*FU1*, *FU2*, *FU3*) (Air-radar groups are modeled as part of environment). Each component has several communication ports. For example, component C2C has three input ports: C2C.R1, C2C.R2 and C2C.R3, and three output ports: C2C.S1, C2C.S2 and C2C.S3. Through these ports the components communicate with each other. The interpretation of ports and transitions appeared in Figure 3 can be found in [6].

3.3 Architectural Constraints Imposed on the C2 System

As the next step, we now formalize the timing-critical system constraints of the C2 system described in Section 4.1 using RTCTL formulae based on the communication ports of the top-level architecture shown in Figure 3. First, requirement (R1) limits the message transferring time delays between input ports SYS.Ri of components SCi, and output ports SYS.Fi of components FUi, i = 1, 2, 3, respectively, so it is naturally specified by the following three path constraints

$$c_i\text{: SYS.Ri} \rightarrow AF^{\leq 45}\,\text{SYS.Fi}, \quad i = 1, 2, 3.$$

The next requirement, R2, in fact bounds the time taken from requests being sent from output port SCi.SM to replies arriving at input ports SCi.RI, i = 1, 2, 3, respectively, so it is naturally specified by the following three environment constraints

$$c_{i+3}\text{: SCi.SM} \rightarrow AF^{\leq 20}\,\text{SCi.RI}, \quad i = 1, 2, 3.$$

The last requirement, R3, also puts a limitation on the message transferring time delay for component C2C between three input ports C2C.Ri and three output ports C2C.Si, i = 1, 2, 3, so it is naturally specified by the following component constraint

$$c_7\text{: C2C.R1} \wedge \text{C2C.R2} \wedge \text{C2C.R3} \rightarrow AF^{\leq 22}\,(\text{C2C.S1} \wedge \text{C2C.S2} \wedge \text{C2C.S3}).$$

The importance of this step 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. Consequently, we gain a clear understanding about what role each subsystem plays in satisfying the global requirements. This also eases analysis as now we deal with specific conditions imposed on each part of the system as opposed to requirement on the system as a whole. This feature becomes increasingly important as we approach more detailed levels of architectural design where the association between a low-level system component and a system-wide requirement becomes much harder to grasp. 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.

3.4 Component Operational Models

With the guidance of the constraints imposed on the system components, we now turn to the internal representation of each component, i.e. how the component satisfy the constraints imposed on it. Figures 4-6 show the TPN models of the C2 Center, the Sub-Center 1, and the Fire Unit 1, respectively. The legend for these models can be found in [6].

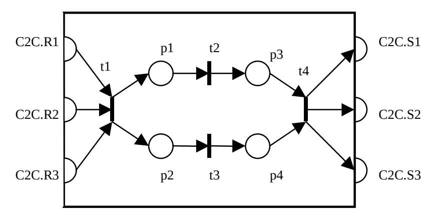

Figure 4. The component operational model of *C2 Center.*
Firing times: t1: [1, 2], t2: [3, 5], t3: [3, 5], t4: [5, 6].

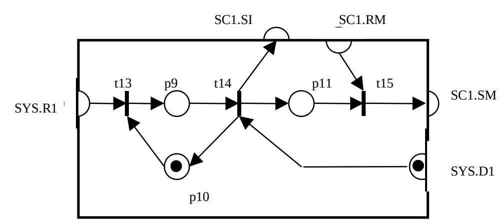

Figure 5. The component operational model of *Sub-Center I.*
Firing times: t13: [2, 3], t14: [1, 2], t15: [4, 6].

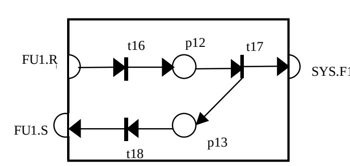

Figure 6. The component operational model of *Fire Unit I.*
Firing times: t16: [4, 6], t17: [1, 2], t18: [5, 7].

3.5 Constraint-Driven Compositional Verification of the C2 System

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 dimensional incremental verification technique that helps achieve these benefits [8]. In this section, we briefly describe this constraint-driven verification method, and illustrate its application on the anti-air C2 system.

A. Constraint-Driven Incremental Verification

Horizontally at each level of design, our algorithm works in two stages. In the first stage, we order the components so that the timing behavior of any component depends only on the components, which are lower in this order. This stage assumes that the underlying dependency-graph of components is acyclic. In the next stage, we consider components one at a time in the order computed in the previous stage. Every component is verified against its component constraints and then reduced to a simple TPN of constant size that preserves the ports and all the time-dependent properties of the component as specified by the constraints, and then composing these results to deduce system-wide properties as specified by environmental constraints and path constraints at that level. In [18], we introduced a set of *component-level reduction rules* to support such closure. It should be pointed out that unlike other reduction methods of TPN, such as given in [17], which work at individual transition level, our reduction rules are constraint-oriented and work at component level. Consequently, our method works at a much coarser level than other reduction rules as given in [17], and we need fewer applications of our rules to reduce the size of the model being analyzed.

Vertically, our method for architectural refinement and incremental verification across design levels consists of three basic elements: 1) We automatically derive constraints for the lower-level design that is consistent with the higher-level constraints. 2) The lower-level design is verified using the technique developed for horizontal verification. 3) The lower-level design is plugged into the parent-level architecture to form a more detailed architectural model. In fact, when we refine a component into a sub-architecture, the RAS model mandates that it inherit all the ports from the higher-level component to ensure interface consistency. During the verification of the higher-level design, we compute time delays for certain port-pairs. Correctness of the high-level design is contingent on the values of these time delays. So for the lower level design, we add these delays as constraints that must be satisfied. As long as these constraints are satisfied, we can ensure that the lower-level designs are consistent with the higher-level design.

B. Verification of the RAS Model of the C2 System

We now consider the incremental verification of the C2 system architectural model against their time-critical system requirements using the algorithm briefly described in Section 3.5(A). The C2 system consists of three types of components: C2 Center (C2C), three Sub-Centers (SC1, SC2 and SC3) and three Fire Units (FU1, FU2 and FU3). Clearly, these components can be ordered based on their dependency relationship as follows:

C2C, FU1, FU2, FU3, SC1, SC2, SC3.

Applying reduction rules [18] to component C2 Center and Fire Units results in Figure 7(a) and (b). Applying simple reachability analysis [4] gives the static firing time intervals of transitions in Figure 7(a) and (b) as

$SI(t_{(C2C.R,\ C2C.S)}) = [9, 13]$,

$SI(t_{(FU1.R,\ SYS.R1)}) = [5, 8]$, and

$SI(t_{(FU1.R,\ FU1.S)}) = [10, 15]$.

So the constraint c_7 is verified. Next, we consider the three sub-centers. Due to assumption (A1), if constraints c_1 and c_4 are verified, then c_2, c_3, c_5 and c_6 can be also be inferred. Furthermore, due to assumption (A2), we can use Figure 7(c), which shows Sub-Center 1 and its interaction with simplified C2 Center and Fire Unit 1, to verify constraints c_1 and c_4. Based on Figure 7(c), we obtain

$SI(t_{(SYS.R1,\ SYS1.F1)}) = [24, 35]$, and

$SI(t_{(SC1.SM,\ SC1.RI)}) = [12, 17]$.

These values imply that c_4 and c_7 are satisfied.

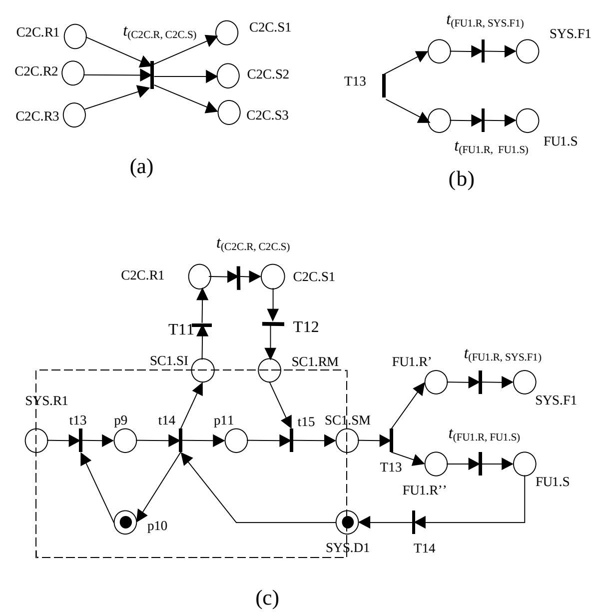

Figure 7. (a) Reduction of component C2 Center.
(b) Reduction of component Fire Unit 1.
(c) The simple but equivalent model of the system.

Therefore, all constraints have been verified.

C. *Discussion*

If, during verification, a timing constraint is found to be violated by the model of architectural design, there are two ways to regain consistency between the system design and requirements:

(1) Modify design model. In other words, redesign the system by modifying the component model(s) that violate the constraint. The modifications may include changing the firing time interval, removing some transitions, or rearranging the control structure of the component. Each change corresponds to a specific modification on the realization and semantics of the system.

(2) Adjust architectural constraints. After the early version design of a system has been finished, we often find that some constraints are too loose, while some constraints are too strict. Such an unbalance assignment of constraints on different functional blocks (components) of the system often leads to the design violating those strict constraints. In this case, a proper adjustment on constraints is necessary.

In either case, because the RAS model tightly integrates architectural constraints with every components of the system at every design level, it is easier to make such adjustments.

For example, suppose that the component constraint c_7 is:

$$c_7\text{: C2C.R1} \wedge \text{C2C.R2} \wedge \text{C2C.R3} \rightarrow AF^{\leq 12}(\text{C2C.S1} \wedge \text{C2C.S2} \wedge \text{C2C.S3})$$

It indicates the whole processing time of component C2 Center is required to be less than or equal to 12 time units instead of 22 time units as given previously. Then we may find that the component C2 Center shown in Figure 4 violates the constraint. From the viewpoint of design, the simplest way to eliminate the violation is to reduce the upper bounds of the firing time intervals of some transitions. For example, we can change the firing time interval of transition t_4 to be [3, 4], which will ensure the consistency between the design and the requirement. As a tradeoff, however, we have to increase the processing speed of the top command seat by over 33%.

3.6 Support for Incremental Design of the C2 System

In this section, we further discuss the incremental nature of RAS in the architectural modeling of the C2 systems. We show how to refine the C2 system architecture by incrementally replace component models with more detailed architectural design. Such an ability is necessary to make a modeling approach scale up.

The RAS model of C2 system built in Sections 3.2 through 3.4 is in fact a high aggregated model. As the design proceeds, we need to introduce more details of the system to the model, which results in the refinement of components described in Figure 3. For example, component *FU1* can be refined into a sub-architecture composed of two low-level components, Engagement Control (*EC*) and Damage Assessment (*DA*), as shown in Figure 8. Note that the refined model inherits all ports of component *FU1* as its external ports. But similarly we must make sure the refined model satisfies all constraints that *FU1* suffers from. From the RAS model we know that component *FUI* is subjected to two constraints, namely c_1 and c_4. Therefore, this sub-architecture design must satisfy c_1 and c_4.

Denote by D_(A, B) the time delay of message transferring between ports A and B. During the high-level verification conducted in Section 3.5(B), we concluded that

$$D\_(\text{FU1.R, SYS.F1}) = SI(t_{(\text{FU1.R, SYS.R1})}) = [5, 8], \text{ and}$$
$$D\_(\text{FU1.R, FU1.S}) = SI(t_{(\text{FU1.R, SYS.R1})}) = [10, 15],$$

which enabled us to prove the constraints c_1 and c_4 for the high-level design. Hence, the low-level model needs to satisfy the following two constraints:

(1) $\text{FU1.R} \rightarrow A(G^{\leq 5} \neg \text{SYS1.F1} \wedge F^{\leq 8} \text{SYS1.F1})$, and

(2) $\text{FU1.R} \rightarrow A(G^{\leq 10} \neg \text{FU1.S} \wedge F^{\leq 15} \text{FU1.S})$.

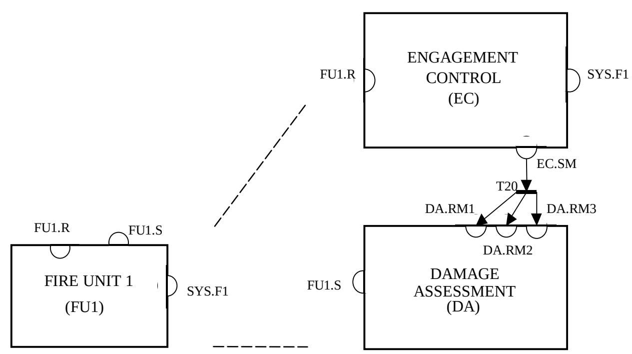

Figure 8. Refinement of Fire Unit 1.

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. We have shown that, through its application to architectural modeling and analysis of a command and control system, the RAS approach can both enhance the dependability of design and improve the usability and scalability of formal techniques in complex system development.

References

[1] Army regulation 5-11, *Army model and simulation management program*, 1992.

[2] M. Athans, Command and control (C2) theory: a challenge to control science. *IEEE Transactions on Automatic Control*, AC-32 (4):286-293, 1987.

[3] R. Bastide, C. Blanc and P. Palanque, Cooperative objects: A concurrent Petri-net based, objected-oriented language. *Procedings of the IEEE International Conference on System, Man and Cybernetics*, 3:286-291, 1993.

[4] B. Berthomieu and M. Diaz, Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3), 1991.

[5] G. Bucci and E. Member, Compositional validation of time-critical systems using communicating time Petri nets, *IEEE Transactions on Software Engineering*, 21(12): 969-992, 1995.

[6] Y. Deng, J. Wang, and X. He, A real-time architectural specification model with application in command and control systems, *Technical Report*, School of Computer Science, Florida International University, 1998.

[7] Y. Deng, J. Wang, and R. Sinha, Integrated architecture modeling of real-time concurrent systems with applications in FMS, *Proceedings of the 10th International Conference on Software Engineering and Knowledge Engineering*, San Francisco Bay, USA June 18-20, 1998.

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

[9] 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.

[10] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi and R. Sherman, M. Trachtenbrot. Statemate: a working environment for the development of complex reactive systems, *IEEE Transactions on Software Engineering*, 16:403-414, 1990.

[11] 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.

[12] H.P. Hillion. Performance evaluation of decision making organizations using timed Petri nets, M.S. thesis, M.I.T., Cambridge, 1983.

[13] Y. Lee and S. Park. Opnets: An object-oriented high-level Petri net model for real-time systems. *Journal of Systems & Software*, SE-13(3):69-86, 1993.

[14] A.H. Levis, N. Moray, and B. Hu, Task decomposition and allocation problems and discrete event systems, *Automatica*, 30(2), 203-216, 1994.

[15] D.K. Pace, Naval modeling and simulation verification, validation, and accredition, *Proceedings of the 1993 Winter Simulation Conference*, 1077-1080, 1993.

[16] A. Pnueli, The temporal logic of programs, *Proceedings of the 18th Annual IEEE Symposium on Fundations of Computer Science*, 46-57, 1977.

[17] R. Sloan and U. Buy, Reduction rules for time Petri nets, *Acta Informatica*, 33:687-706, 1996.

[18] J. Wang and Y. Deng, Component-level reduction rules for time Petri nets with application in C2 systems, *Proceedings of the IEEE Conference on System, Man and Cybernetics*, San Diego, California, USA, October 1998.

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

[20] A.K. Zaid and A.H. Levis, Validation and verification of decision making rules, *Automatica*, 33(2), 155-169, 1997.