ELSEVIER

Information and Software Technology 41 (1999) 451–467

INFORMATION AND SOFTWARE TECHNOLOGY

Introducing software architecture specification and analysis in SAM through an example[☆]

J. Wang[a], X. He[b,*], Y. Deng[a]

[a]School of Computer Science, Florida International University, University Park, Miami, FL 33199, USA
[b]Department of Computer Science, North Dakota State University, Fargo, ND 58105, USA

Received 2 December 1998; received in revised form 30 December 1998; accepted 27 January 1999

Abstract

Software architecture study has become one of the most active research areas in software engineering in the recent years. Although there have been many published results on specification and analysis method of software architectures, information on sound systematic methodology for modeling and analyzing software architectures is lacking. In this article, we present a formal systematic software architecture specification and analysis methodology called SAM and show how to apply SAM to specify a command control (C2) system and to analyze its real-time constraints. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Software architecture; Formal methods; Real-time concurrent systems; Time Petri nets; Real-time computational tree logic; Command and control systems

1. Introduction

Software architecture study has become one of the most active research areas in software engineering in recent years [1]. Although there have been many published results on specification and analysis methods of software architectures [2,3], information on sound systematic methodology for modeling and analyzing software architectures is lacking. In this article, we present a formal systematic software architecture specification and analysis methodology (called SAM) through a command control (C2 system). The goal of this article is twofold. Firstly, we introduce SAM in a light-weight fashion, and secondly, we demonstrate the specification and analysis of the software architecture of a typical C2 system. As pointed out in [1], the examination of significant architectural case studies can greatly help in defining the field of software architecture [4,5].

As our emphasis of this article is to demonstrate SAM through a C2 system, we keep the discussion light on the foundation and methodology of SAM and only present the features of SAM relevant to the C2 system.

SAM has been developed with the goal to define software architectures and their properties precisely and to facilitate their formal analysis, and thus use formal methods [6–8] as its foundation. Sharing the similar views with several other formal software architecture approaches (Rapide [2] and CHAM [3]), we see the great importance of an executable software architecture specification and thus use time Petri nets (TPN) [9–11] as one of the foundations of the underlying model of SAM. However, it is well known that the property-oriented formal methods are suitable for formal verification [12,13], and hence we use real-time computational tree logic [14] as the other formal foundation of SAM model to explicitly represent and reason about the architectural properties at a high level. Although dual formal methods have been used in system specification and verification (for example, integrating Petri nets with temporal logic [15–17] and their advantages have been widely recognized in the recent years [18]; our use of dual complimentary formal notations in a unified framework for software architecture study is distinct and unique.

Another major goal of SAM is to support the scalability of software architecture specification and analysis. SAM supports the scalability through hierarchical architectural decomposition. Although the concept of hierarchical

☆ This work was supported in part by the NSF under grant No. HDR-9707076, by the Air Force Office of Scientific Research under grant No. F49620-96-1-0221, by the Army Research Office under grant No. DAAG55-98-1-0428, and by the Office of Naval Research under grant No. N00014-98-1-0591. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied by the aforementioned Agencies.

* Corresponding author. Tel.: + 1-305-346-2744; fax: + 1-305-348-3549.

E-mail addresses: wangji@cs.fiu.edu (J. Wang); hex@plains.nodak.edu (X. He); deng@cs.fiu.edu (Y. Deng)

architectural decomposition is not new and has been used in several other approaches (Unicon [19] and Rapide [2]), SAM supports both hierarchical software architecture specification and analysis. In SAM, a software architecture is modeled as a multi-leveled composition of subsystems (components and connectors specified in TPN), together with the specification of system requirements (specified in RTCTL). Each component (connector) has its own property specification (in RTCTL) and behavior model (in TPN). The specification and analysis of each component (connector) is done separately. The propagation of the requirements goes in lockstep with the refinement of system design, and serves as the design goals of the subsystems in the refinement. Doing so, SAM lays a foundation for an incremental process of architectural refinement and analysis that is both scalable and flexible. Horizontally at each design level, a system model can be constructed and analyzed compositionally. Vertically across design levels, lower level (interface specification conforming) sub-architecture can be built and analyzed incrementally and recursively.

2. Overview of SAM

In SAM, system architecture is specified by a set-theoretical recursive definition such that multi-layered compositions and hierarchical system refinements can be defined and analyzed separately. Horizontally at any given abstraction level, the model captures not only the operational property (modules—components and connectors, and their composition) but also the descriptive architectural requirements (called *architectural constraints* in SAM) that each of the modules and their composition must satisfy at every design level. Such a strong correlation between system design and system requirements systematically maintains the integrity of design against requirements. Vertically, there is a mapping between any consecutive design levels. Particularly, the propagation of the requirements goes in lockstep with the refinement of system design, and serves as the design goals of the subsystems in the refinement, as well as the target of verification. During architectural design, every decision can be traced backward to the requirements, and conversely every requirement can be traceable forward to architectural decisions and designs. Consequently, design traceability and conformity as defined earlier is maintained while avoiding ad hoc, accidental design and unjustified efforts.

SAM also lays a foundation for a goal-directed, incremental process of architectural refinement and analysis that is both scalable and flexible. System components in SAM interact with environment only via their communication interface. Moreover, the requirement specification deals only with the interface with internal states of the components encapsulated. This property has the following implications. Horizontally, at each design level, a system model can be constructed and analyzed compositionally. Vertically across design levels, lower level sub-architecture can be built and analyzed incrementally and recursively.

Both top–down and bottom–up system development approaches are supported. The top–down approach is used to develop a software architecture specification by decomposing a system specification into specifications of components and connectors and by refining a higher level component into a set of related sub-components and connectors at a lower level. The bottom–up approach is used to develop an architecture specification by composing existing specifications of components and connectors and by abstracting a set of related components and connectors into a higher level component. No matter what approach is adopted, the consistency between critical system requirements and the system design at every step is systematically enforced by architectural integrity rules within the SAM model from beginning to end. Some examples of these rules are:

- All architectural constraints must be consistent at any design level, that is, the satisfaction of one constraint must not lead to the violation of any other constraints.
- The behavior model for a component or sub-architecture at a given level must satisfy the corresponding architectural constraints imposed on the component or sub-architecture.
- A sub-architecture at design level $k + 1$ must inherit all the ports associated with its corresponding component at level k (Fig. 1).
- A sub-architecture at design level $k + 1$ must conform to all constraints which its corresponding component at level k is subject to (Fig. 1).

The central part of SAM is its specification model, illustrated in Fig. 1. A SAM model consists of three basic elements: *component models* and their specifications (also called *component constraints), connector models* and their specifications (also called *connector constraints*), and *architectural constraints* organized into multi-design levels. The component models describe the behavior and communication interfaces (called *ports*) of the components. The connectors specify how the components interact with each other and, in turn, form the *composition model*.

The architectural constraints define requirements imposed on the components and connectors, and are divided into *environment constraints* of each individual module (component or connector) and *composition constraints* involving multiple components. All connectors 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.

The component ports also provide the linkage between the operational design (components and connectors) and the descriptive architectural constraints. In particular, system constraints are specified by temporal formulas defined over ports, which constitute the alphabet of the formulas. All constraints are specified using ports only, no internal

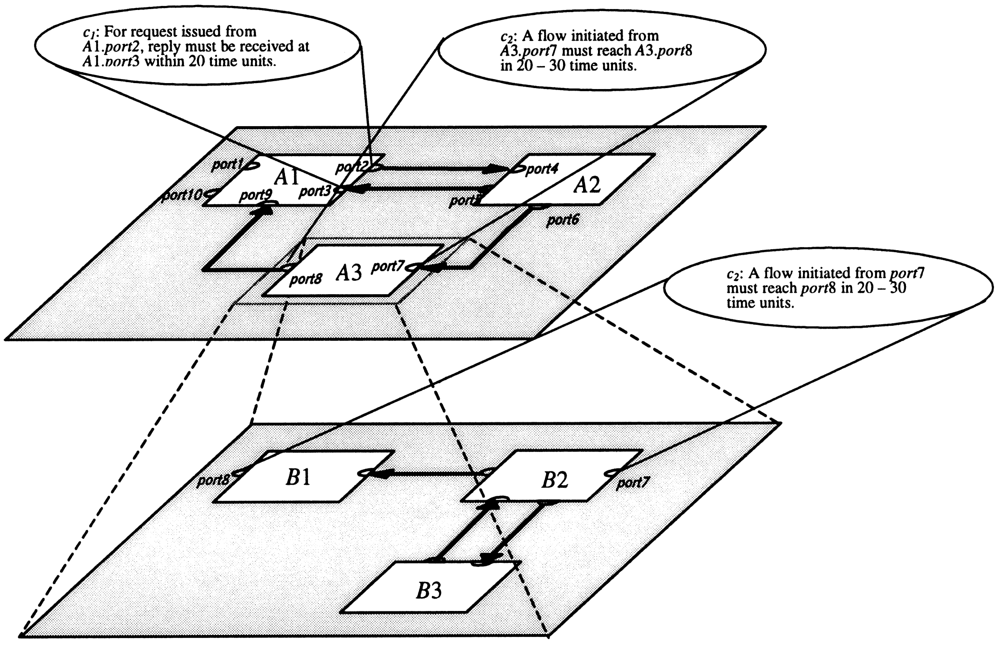
c1: For request issued from A1.port2, reply must be received at A1.port3 within 20 time units.
c2: A flow initiated from A3.port7 must reach A3.port8 in 20 – 30 time units.
c2: A flow initiated from port7 must reach port8 in 20 – 30 time units.
port1
port2
port3
port4
port5
port6
port7
port8
port9
port10
A1
A2
A3
B1
B2
B3
port8
port7

Fig. 1. Framework of the SAM model.

information about the components and connectors is revealed. This arrangement is critical to achieve the goals of incremental modeling and analysis described earlier, as it ensures that the component and connector designs can be treated as black-boxes in the construction, understanding and analysis of system's architecture.

The strong linkage is maintained during the whole process of design. As shown in Fig. 1, when component A3 is decomposed into a sub-architecture composed of components B1, B2 and B3, all the ports of A3, i.e. *port7* and *port8*, are inherited in the sub-architecture, and all constraints defined on the component, e.g. constraint c_2, are inherited in the sub-architecture. This ensures interface consistency between design models at different levels.

Formally, an SAM model consists of a set of compositions C (a composition may correspond to a design level, or the concept of sub-architecture) and a hierarchical mapping h.

SAM = (C, h), where

1. $C = \{C_1, C_2,\ldots, C_k\}$, and $C_i = \{Cm, Cn, Cs\}$ for each C_i, where

   (i) Cm is a set of components. For each $Cm_j \in Cm, Cm_j = (Sm_j, Bm_j)$, where

   - Sm_j is a property specification (component constraints) of component Cm_j. It is defined by a set of RTCTL formulae.
   - Bm_j is the behavior model of component Cm_j. It is defined by a TPN. Let

   $$Cm_jPORT = \{p | p \in Cm_j.P \wedge \cdot p \cap Cm_j.T = \emptyset \wedge p \cdot \cap Cm_j.T = \emptyset\}.$$

   $Cm_j.PORT$ is the set of ports of component Cm_j, which defines the interface of the component (any place p with an empty input pre-set, $\cdot p \cap Cm_j.T = \emptyset$, is an input port, and any place p with an empty post-set, $p \cdot \cap Cm_j.T = \emptyset$, is an output port). In addition there is no common node between any two components, i.e. for $\forall Cm_j, Cm_k \in C_i.Cm$,

   $$Cm_j.P \cap Cm_k.P = \emptyset, \qquad Cm_j.T \cap Cm_k.T = \emptyset.$$

   For any constraint c, denoted by $c.PORT$ the set of ports which are used as atomic propositions of c. Thus for each $c \in Sm_j$, the following holds

   $$c.PORT \subseteq Cm_j.PORT.$$

   That is, a component constraint only uses ports that belong to the component.

   (ii) Cn is a set of connectors. For each $Cn_j \in Cn, Cn_j = (Sn_j, Bn_j)$, where

   - Sn_j is a property specification (connector constraints) of connector Cn_j. It represents the requirements on its functionality and is defined by a set of RTCTL formulae.
   - Bn_j is the behavior model of connector Cn_j. It is defined by a TPN such that

   $$\bigcup_{Cn_j \in Cn} Cn_j.P \cap \left(\bigcup_{Cm_j \in Cm} Cm_j.P \backslash Cm_j.PORT \right) = \emptyset,$$

   $$\bigcup_{Cn_j \in Cn} Cn_jT \cap \left(\bigcup_{Cm_j \in Cm} Cm_j.T \right) = \emptyset.$$

The aforementioned conditions require that a connector cannot use any internal nodes of a component as its own nodes. Similarly, for each $c \in Sn_j$, the following condition holds:

$c.PORT \subseteq Cn_j.PORT$.

The overall behavior (TPN) model of composition C_i is defined by the union of all the component and the connector models within it:

$$C_i.P = \left(\bigcup_{Cm_j \in Cm} Cm_j.P \right) \cup \bigcup_{Cn_j \in Cn} Cn_j.P,$$

$$C_i.T = \left(\bigcup_{Cm_j \in Cm} Cm_j.T \right) \cup \bigcup_{Cn_j \in Cn} Cn_j.T.$$

Let

$$C_i.PORT\_EXT = \left\{ p \middle| p \in \bigcup_{Cm_j \in Cm} Cm_j.PORT \wedge \cdot p \cap C_i.T = \emptyset \wedge p \cdot \cap C_i.T = \emptyset \right\},$$

where $C_i.PORT\_EXT$ is the set of ports that are not used by any connector. Ports in $C_i.PORT\_EXT$ are called *external* ports of C_i.

(iii) Cs is a set of architectural constraints. Each $Cs_j \in Cs$ is an RTCTL formula and it only uses ports as its atomic propositions. Similar to component constraints and connector constraints, the atomic proposition is true at the moment τ if:

- marking transition happens at τ, and
- the port contains a token in the new marking. In the temporal structure $\Sigma = (S, R, L), S = (M, \varphi_M)$ where M is a TPN marking, and φ_M is the global time when the TPN enters M; R is a binary relation on S, which is indicated by firing transitions; and L is a mapping: $(M, \varphi_M) \rightarrow C_i.PORT$. In addition, the following condition is enforced:

$$\bigwedge_{c \in Sm_i \cup Sn_i \cup Cs} c. \quad (1)$$

2. $\forall C_i \in C, \forall Cm_l \in C_i.C_m, h : Cm_l \rightarrow C_j,\ j \neq i$, such that

- $Cm_l.PORT = C_j.PORT\_EXT$, (2)
- $Cm_l.Sm_l \subseteq C_j.Cs$. (3)

In the aforementioned definition, expression (1) states that all constraints should be consistent with each other, and it establishes the (*horizontal*) *constraint/specification consistency condition*. Eq. (2) states that when refining a component into a sub-architecture, the sub-architecture must inherit all ports of the component as all its external ports, and it establishes the *structural consistency condition*. Eq. (3) states that when refining a component into a sub-architecture, the sub-architecture must conform to all constraints/specifications which the component are subject to (*behavioral consistency*). Such a consistency ensures that the system requirements are met in every step of the design process. Eqs. (2) and (3) together give the *vertical* (*interface*) *consistency conditions*.

A brief introduction to TPN and RTCTL is given in Appendix A.

3. Specifying the software architecture of a C2 system

A C2 system is a distributed modularized system. It achieves mission success by executing a set of generally accepted C2 functions in an asynchronous manner. These functions include [22]:

1. *Threat Detection*: to collect threats data from various sensors.
2. *Discrimination*: to distinguish real threats from decoys. Sensor cueing, scheduling, and control are also an integral part of this function.
3. *Identification and Tracking*: to establish the identity information of threats.
4. *Threatening Assessment*: to quantify the impact of each threat.
5. *Battle Planning*: to make decisions on how to deal with the identified threat, including contingency planning.
6. *Weapon-to-Target Assignment* (or *Fire Assignment*): to determine the deployment of weapon systems to each threat, including the assignment of any other necessary resources such as sensor and communication equipment.
7. *Engagement Control*: to timely execute the decisions made in (5) and (6).
8. *Damage Assessment*: to determine the outcome of the engagement, i.e. whether a particular target has been destroyed or not.

A typical structure of a tactic anti-air C2 system with two levels of command and control centers is shown in Fig. 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.

3.1. Requirements of the C2 system

A C2 system is a typical real-time system, and over delay in execution of any of its functions may incur severe results. In this example, 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 original user requirements

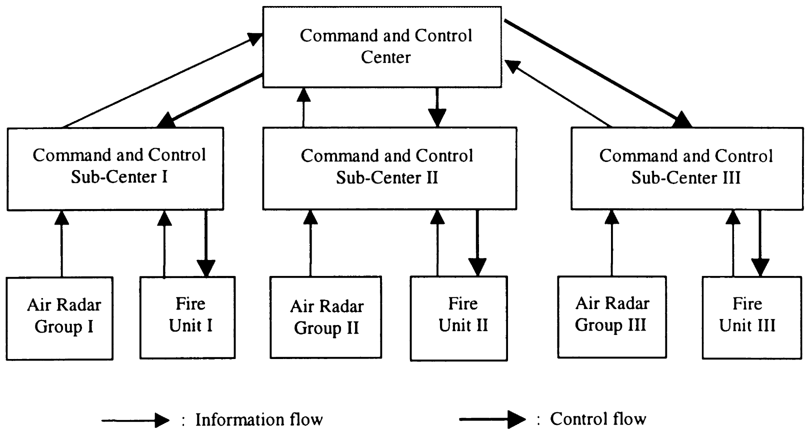
Command and Control Center
Command and Control Sub-Center I
Command and Control Sub-Center II
Command and Control Sub-Center III
Air Radar Group I
Fire Unit I
Air Radar Group II
Fire Unit II
Air Radar Group III
Fire Unit III
: Information flow
: Control flow

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

include:

(R1) The system reaction time, i.e. the time delay from a enemy intelligence message being received by any sub-center to a fire command against the enemy being issued by a corresponding fire unit, must be less or equal to 45 time units.[1]

(R2) As 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 detailed firing assignment scheme made by a sub-center to the result of the damage assessment referred to the execution of this scheme received by the same sub-center must be less or equal to 20 time units.

(R3) As 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.

(R4) A C2 system is a real-time information processing system. Obtaining targets' information timely and continuously is extremely important to win a war. This is captured by the requirement that each Radar Group outputs targets' information periodically at a period of 40 time units.

In order to simplify analysis, we also assume that all the three sub-centers have identical topology and timing properties, and all the three fire units have identical topology and timing properties.

3.2. Formalizing the C2 requirements

As pointed out in Section 2, all SAM architectural constraints or specifications are formally defined on ports, so we now introduce the top-level architecture of the system, which also capture the profile of the SAM model of the C2 system. As shown in Fig. 4, there are total 10 components: a C2 center (C2C), three sub-centers (SC1, SC2, SC3), three radar groups (RG1, RG2, RG3), and three fire units (FU1, FU2, FU3), and nine connectors in the system. 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 Table 1 presents the legend for Fig. 3. Owing to the symmetry, we only describe the ports and transitions for radar group I, C2 Center, sub-center I, and fire unit I.

Now we can formalize the timing-critical system requirements of the C2 system described in Section 3.1 using RTCTL formulae based on the communication ports of the top-level architecture shown in Fig. 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 composition constraint:[2]

$$\mathrm{pc}_1 : \mathrm{SYS.R1} \wedge \mathrm{SYS.R2} \wedge \mathrm{SYS.R3} \rightarrow AF^{\leq 45}(\mathrm{SYS.F1} \wedge \mathrm{SYS.F2} \wedge \mathrm{SYS.F3}).$$

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, $i = 1, 2, 3$, so it is naturally specified by the following three environment constraints:

$$\mathrm{ec}_i : \mathrm{SC}i.\mathrm{SM} \rightarrow \mathrm{AF}^{\leq 20}\mathrm{SC}i.\mathrm{RI}, \quad i = 1,2,3.$$

[1] Notice that the timing data in this example is artificial and does not reflect actual data in a real system.

[2] In RTCTL formulae there are four temporal operators: Fp ("eventually p"), Gp ("always p"), Xp ("nexttime p"), and pUq ("p until q"), and two path quantifer: A ("for all futures") and E ("for all futures"). Formula $p \rightarrow \mathrm{AF}^{\leq n}q$ says that if p is true, then q will be guaranteed true within next n time units.

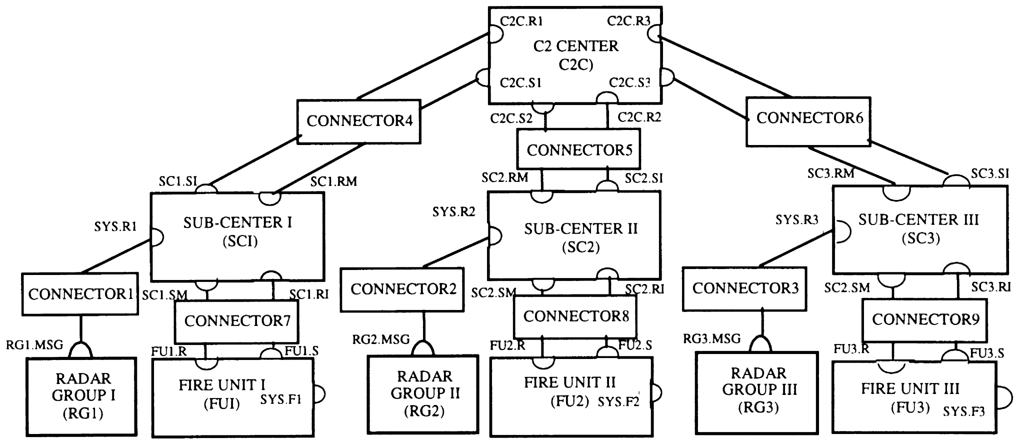

Fig. 3. Architecture of the tactic anti-air C2 system.

The third 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 constraints:

$$cc_1 : \text{C2C.R1} \wedge \text{C2C.R2} \wedge \text{C2C.R3}$$

$$\rightarrow AF^{\leq 22}(\text{C2C.S1} \wedge \text{C2C.S2} \wedge \text{C2C.S3}).$$

The last requirement, R4, is on the output rate of components RGi, $i = 1, 2, 3$. It is specified by the following component constraint:

$$cc_{i+1} : \text{RG}i\text{.MSG} \rightarrow A(G^{\leq 30} \neg\, \text{RG}i\text{.MSG})$$

$$\wedge(F^{\leq 40}\text{RG}i\text{.MSG}), i = 1, 2, 3.$$

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.

Table 1
Legends of partial ports in Fig. 3

Port	Type	Description
RG1.MSG	Output	Radar group 1 ready to send target message to sub-center I
SYS.R1	Input	A message from air radar group I arrived
SYS.F1	Output	A combat command to fire unit I sent
SC1.SI	Output	Sub-center I ready to send intelligence to command and control center
SC1.RM	Input	Sub-center I received command from command and control center
SC1.SM	Output	Sub-center I ready to send command to fire unit I
SC1.RI	Input	Sub-center I received result of damage assessment from fire unit
FU1.R	Input	Fire unit I received command from sub-center I
FU1.S	Output	Fire unit I ready to send result of damage assessment to sub-center I
C2C.R1	Input	Command and control center received message from sub-center I
C2C.S1	Output	Command and control center ready to send command to sub-center I

3.3. Defining component and connector specifications

We need to derive a set of intermediate constraints or specifications to guide the design of each component and each connector. The task of imposing original timing parameters on the functional components is a complex one, and it mandates some careful engineering. As the original constraints allow many possibilities for the intermediate constraints, and engineers make what they consider to be a rational selection, as the design goes on, they may find it is necessary to make another selection [23].

As an important part of our SAM modeling and design, the deriving of intermediate constraints should satisfy (horizontal) constraint consistency condition [Eq. (1)], structural consistency condition [Eq. (2)] and behavioral consistency condition [Eq. (3)]. Moreover, in order to design a component/connector independent of the rest of a system, it is desirable that there is a set of constraints which completely describe what properties of the component are expected by its environment, and what properties the component/connector expects from its environment. Such a property

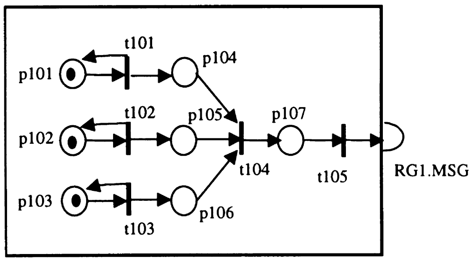

Fig. 4. The operational model of component radar group I.

Table 2
Legend for Fig. 4

Place	Description	
*p*101, *p*102, *p*103	Radar ready to sense air targets	
*p*104, *p*105, *p*106	Radar data for fusion	
Transition	**Description**	**Firing interval**
*t*101, *t*102, *t*103	Radar senses	[30,30]
*t*104	Processor fuses data	[2,4]
*t*105	Processor codes fused data	[1,2]

is extremely expected when we conduct designs of complex real-time systems. For our example, we may get the following intermediate constraints:

$$\mathrm{cc}_{i+4}: \mathrm{SYS.R}i \wedge \mathrm{SC}i\mathrm{.RI} \rightarrow \mathrm{AF}^{\leq 8}\mathrm{SC}i\mathrm{.SI}, \quad i = 1,2,3$$

(component constraints, for SC1, SC2 and SC3, respectively),

$$\mathrm{ec}_4: \mathrm{SC1.SI} \wedge \mathrm{SC2.SI} \wedge \mathrm{SC3.SI}$$

$$\rightarrow \mathrm{AF}^{\leq 25}\mathrm{SC1.RM} \wedge \mathrm{SC2.RM} \wedge \mathrm{SC3.RM},$$

(environment constraint, for SC1, SC2 and SC3),

$$\mathrm{cc}_{i+7}: \mathrm{SC}i\mathrm{.RM} \rightarrow \mathrm{AF}^{\leq 8}\mathrm{SC}i\mathrm{.SM}, \quad i = 1,2,3$$

(component constraints, for SC1, SC2 and SC3, respectively),

$$\mathrm{cc}_{i+10}: \mathrm{FU}i\mathrm{.R} \rightarrow \mathrm{AF}^{\leq 8}\mathrm{SYS.F}i, \quad i = 1,2,3$$

(component constraints, forFU1, FU2 and FU3, respectively),

$$\mathrm{cc}_{i+13}: \mathrm{FU}i\mathrm{.R} \rightarrow \mathrm{AF}^{\leq 16}\mathrm{FU}i\mathrm{.S}, \quad i = 1,2,3$$

(component constraints, for FU1, FU2 and FU3, respectively)

Now ec_i, cc_{i+2}, ec_4 and cc_{i+7} completely describe the timing requirements on component SC*i*, *i* = 1, 2, 3; cc_{i+10} and cc_{i+13} completely describe the timing requirements on component FU*i*, *i* = 1, 2, 3; cc_1 itself completely describes the timing requirements on component C2C; and cc_{i+1} itself completely describes the timing requirements on component RG*i*, *i* = 1, 2, 3.

We also derive intermediate constraints to guide the design of connectors as follows:

$$\mathrm{nc}_i: \mathrm{RG}i\mathrm{.MSG} \rightarrow \mathrm{AF}^{\leq 2}\mathrm{SYS.R}i,$$

$$\text{for connector } i, \quad i = 1,2,3,$$

$$\mathrm{nc}_{i+4}: \mathrm{SC}i\mathrm{.SI} \rightarrow \mathrm{AF}^{\leq 2}\mathrm{C2C.R}i \wedge \mathrm{C2C.S}i \rightarrow \mathrm{AF}^{\leq 2}\mathrm{SC}i\mathrm{.RM},$$

$$\text{for connector } i+3, \ i = 1,2,3,$$

$$\mathrm{nc}_{i+7}: \mathrm{SC}i\mathrm{.SM} \rightarrow \mathrm{AF}^{\leq 2}\mathrm{FU}i\mathrm{.R} \wedge \mathrm{FU}i\mathrm{.S} \rightarrow \mathrm{AF}^{\leq 2}\mathrm{SC}i\mathrm{.RI},$$

$$\text{for connector } i+6, \ i = 1,2,3.$$

After this step, we gain a clear understanding about what role each component or connector plays to satisfy 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 to the system as a whole. This feature becomes increasingly important as we approach to 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.

3.4. Constructing component and connector behavior 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. First let us consider the component of radar group 1. Each component of radar group is composed of three air radars and a data processor. The specification is:

1. Each radar periodically senses air targets, and all the three radars share the same periods;
2. The data from the three air radar are fused at the processor;
3. The fused data are coded and sent to its corresponding sub-center.

Fig. 4 shows the component operational model of radar group I. Table 2 gives the description for all the nodes in the figure.

The C2 center is composed of three *seats*: two *intelligence seats*, and one *decision-making seat*. The behavior specification of this component is as follows:

1. The two intelligence seats communicate with the decision-making seat through a common memory.
2. The messages from three sub-centers are dispatched to the two intelligence seats.
3. The two intelligence seats are responsible for performing *fusion* for these messages to achieve an overall situation figure, and then make *threatening assessment*

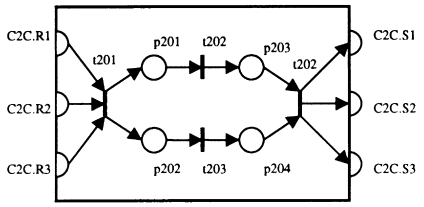

Fig. 5. The component operational model of C2 center.

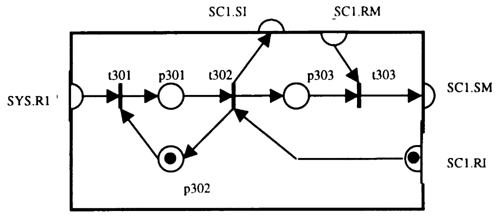

Fig. 6. The component operational model of sub-center I.

Table 3
Legend for Fig. 5

Place	Description	
*p*201, *p*202	Ready for situation assessment	
*p*203, *p*204	Ready for fusion and combat planning	
Transition	**Description**	**Firing interval**
*t*201	Dispatch intelligence message to two staff seats	[1,2]
*t*202, *t*203	Two staff seats conduct situation assessment	[3,5]
*t*204	Top commander seat conducts information fusion and combat planning	[5,6]

Table 4
Legend for Fig. 6

Place	Description	
*p*301	Waiting for evaluation of threat	
*p*302	Intelligence seat available	
*p*303	Waiting for decision-making of fire assignment	
Transition	**Description**	**Firing interval**
*t*301	Sub-center I conducts target discrimination, identification and tracking	[2,3]
*t*302	Sub-center I conducts threatening assessment	[1,2]
*t*303	Sub-center I conducts fire assignment	[4,6]

independently for each target and sends the results to the decision-making seat.

4. The decision-making seat works on a scheme of *battle planning*. The result is sent to the three sub-centers.

The behavior properties of this component are modeled by TPN as shown in Fig. 5. Table 3 gives the description for all the nodes in the figure. Notice that the required synchronization among messages from the three sub-centers has been modeled by transition *t*201, which does not fire until messages from all the three centers have been received.

Now we turn to component sub-center I. Each sub-center is composed of a intelligence seat and a decision-making seat. The behavior specification of this component is as follows:

1. The intelligence seat receives the message from its radar group and conducts *target discrimination*, *identification* and *tracking*, and further conducts *threatening assessment*, then sends the result to the C2 center.
2. After receiving the scheme of *battle planning* from C2 center, the sub-center fuses it with related data in the database again so as to form a detailed scheme of *weapon-to-target assignment*. Further, the results are sent to fire units.

Fig. 6 shows the component operational model of Sub-Center I. Table 4 gives the description for all the nodes in the figure.

Now we avert to component fire unit I. The specification of this component is as follows:

1. When the scheme of *weapon-to-target assignment* arrives from its sub-center, it conducts *engagement control* and sends fire command to weapons.
2. Then, it conducts *damage assessment*, then feedbacks the assessment results to its corresponding sub-center in time.

The component behavior model of fire unit I is shown in Fig. 7. Table 5 gives the description for all the nodes in the figure.

All connectors describe the information transferring among system components, and their behavior models are very simple. Figs. 8–10 show the model of connectors 1, 4, and 7, respectively. All transitions (indicating the information transferring) have the same static firing time interval of [1,2]. Owing to the symmetry, connectors 2 and 3 have the same model as connector 1, so connectors 5 and 6 as connector 4, and connectors 8 and 9 as connector 7.

Once these component and connector behavioral models are constructed, we have a complete system architectural model (at this design level) both syntactically and semantically, which is also executable. To enforce the integrity of the design against system requirements, it is important to verify and/or validate that the components and their interactions satisfy their respective constraints. The degree of assurance that a system design can offer depend directly on ability to verify such a conformance, and the applicability of a modeling and design method depend directly on whether such verification can be done incrementally and systematically. We argue that our approach contributes to

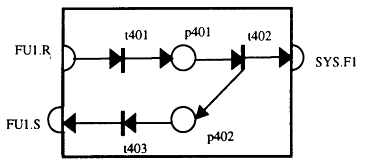

Fig. 7. The component operational model of fire unit I.

Table 5
Legend for Fig. 7

Place	Description	
*p*401	Ready to send fire command	
*p*402	Waiting for damage assessment	
Transition	Description	Firing interval
*t*401	Fire unit I conducts engagement control	[4,6]
*t*402	Fire unit I sends fire command	[1,2]
*t*403	Fire unit I conducts damage assessment	[5,7]

both of these two objectives for the reasons described earlier. The verification method that helps to achieve these objectives will be presented in Section 4.

3.5. Supporting incremental design of the C2 system

In this section, we further discuss the incremental nature of SAM in the architectural modeling of the C2 systems.

As we have seen in Section 3.1, SAM interface provides an information-hiding template for system composition, that is, a specific design of a component can be "plugged-into" the place of the corresponding specification to form a new SAM model. Such a template naturally supports for incremental design of distributed systems.

Two issues are discussed in this section. First, we discuss support for exploring alternative designs, which is essential for complex system design, where design conditions may change over time or repeated experiments are necessary to find a suitable design. Second, we show how to refine the C2 system architecture by incrementally replace component models with more detailed architectural design. Such ability is necessary to make a modeling approach scale up.

3.5.1. Replacing component C2 center with an alternative design

If there are several seats included in a C2 center or C2 sub-center, as a reliable and powerful communication media, a local area network (LAN) is often adopted in engineering. For simplicity, the alternative design of the C2 Center considered here is still composed of two intelligence seats and one decision-making seat, but differently, they are connected with each other through a LAN instead of through a common memory. LAN adopts a CSMA/CD protocol.

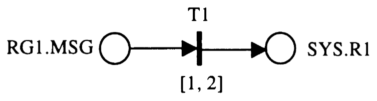

Fig. 8. Model of connector 1.

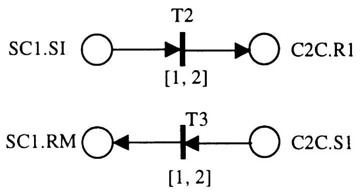

Fig. 9. Model of connector 4.

The control structure for the alternative design of the C2 Center is as follows: The messages from three sub-centers are first grouped and two copies of the grouped message are sent to the two intelligence seats through LAN. The two intelligence seats are responsible for performing *fusion* for these messages to achieve a relatively higher precision and overall situation figure, and then make *threatening assessment* independently for each target and sends the results to the decision-making seat through LAN. The decision-making seat works on a scheme of *battle planning*. The result is sent to the three sub-centers.

The component model for the alternative design of the C2 Center is shown Fig. 11. As we see, this model is in fact a colored TPN. Here we use the colored TPN model is just to give a compact presentation. It does not affect the semantics of the component's interface. Also we assume that the LAN takes the same processing time for messages sent from the two intelligence seats and the decision-making seat in the same processing stage. Table 6 shows the legends of internal places and transitions in this model. Notice that *t*203, *t*204 and *t*208 are immediate transitions, that is, they will fire in 0 time units when they are enabled.

In order to be consistent, this alternative design must satisfy all constraints that the old component design, shown in Fig. 5, satisfies. As pointed out in Section 3.3, constraint cc_1 completely describes the timing requirement of the system on component C2C. Therefore, the alternative design is subjected to constraint

$$\mathrm{C2C.R1} \wedge \mathrm{C2C.R2} \wedge \mathrm{C2C{\cdot}R3} \rightarrow \mathrm{AF}^{\leq 22}(\mathrm{C2C.S1} \wedge \mathrm{C2C.S2} \wedge \mathrm{C2C.S3}).$$

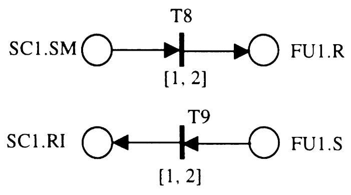

Fig. 10. Model of connector 7.

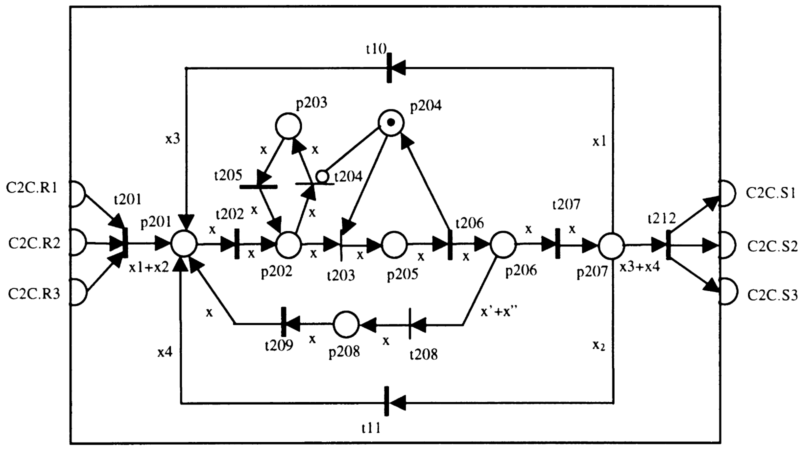

Fig. 11. The internal representation of component C2 center.

Table 6
Legend for Fig. 11

Place	Description	
*p*201	Ready for situation assessment	
*p*202	A station sensed the channel	
*p*203	The channel is busy	
*p*204	The channel is available	
*p*205	The channel is idle	
*p*206	The packet is on transmission	
*p*207	The packet reached the destination station	
*p*208	Collision is detected	
Transition	**Description**	**Firing interval**
*t*201	The messages from sub-centers are grouped and communicated to LAN	[1,2]
*t*202	Sense the channel	[1,1]
*t*203	Transmission begins	—
*t*204	The channel senses busy and begin waiting	—
*t*205	Re-sense the channel	[1,1]
*t*206	Detect collision	[1,1]
*t*207	Transmission ends successfully	[1,1]
*t*208	Collision happens	—
*t*209	Wait for a back-off time	[1,2]
*t*210, *t*211	Two intelligence seats conduct situation assessment	[3,5]
*t*212	Decision-making seat conducts information fusion and combat planning	[5,6]

Then we can use the compositional verification algorithm given in Section 4 to verify the new design.

3.5.2. Refining architectural design incrementally

The SAM model of C2 system built in Sections 3.2–3.4 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 Fig. 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 Figs. 12 and 13 (see Tables 7 and 8 legends). Note that the refined model inherits all ports of component FU1 as its external ports, as enforced by the structural consistency condition. Further, the refined model need to satisfy all constraints that FU1 suffers from, as enforced by the behavioral consistency condition. From the SAM model we know that component FU1 is subject to two constraints, namely cc_{11} and cc_{14}. Therefore, this sub-architecture design must satisfy cc_{11} and cc_{14}.

Table 7
Legends of new ports and transitions in Fig. 12

Port	Description	Type
EC.SM	Ready to send command for collecting firing results	Output
DA.RM1,2,3	Waiting for damage assessment	Input
Transition	**Description**	**Firing interval**
*t*20	Fire unit I sends command of collecting firing results	[1,2]

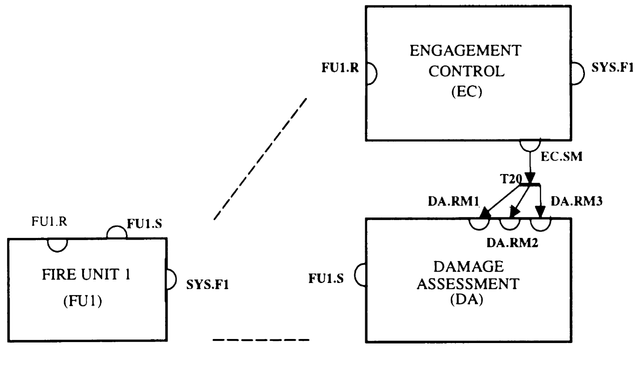

Fig. 12. Refinement of fire unit 1.

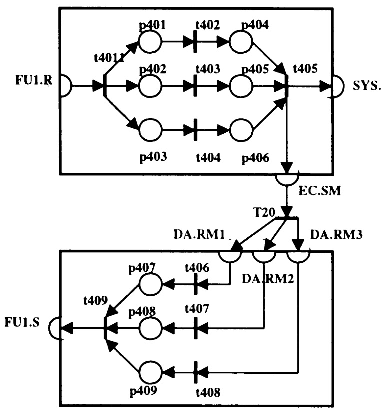

Fig. 13. Internal representation of refined fire unite 1.

Table 8
Legends of places and transitions in Fig. 13

Place	Description	
*p*401, *p*402, *p*403	Ready to send fire command	
*p*404, *p*405, *p*406	Ready for evaluation of shot effectiveness	
*p*407, *p*408, *p*409	Ready for evaluation of shot effectiveness	
Transition	**Description**	**Firing interval**
*t*401	Fire unit I conducts calculation of shot parameters	[4,6]
*t*402, *t*403, *t*404	Weapons (three in total) are aimed at targets	[2,4]
*t*405	Fire unit I fires at targets	[2,4]
*t*406, *t*407, *t*408	Sensors (three in total) get the fire result	[1,2]
*t*409	Fire unit I conducts damage assessment	[3,5]

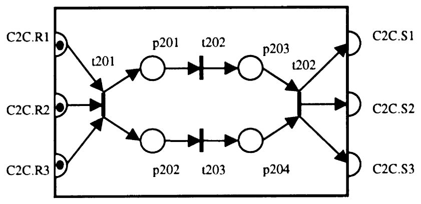

Fig. 14. Behavior model for verifying cc_1.

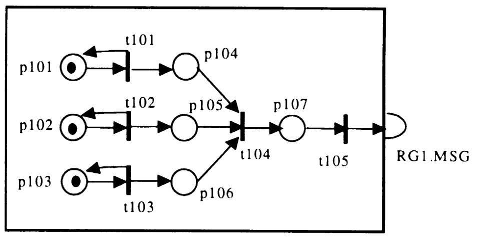

Fig. 15. Behavior model for verifying cc_2.

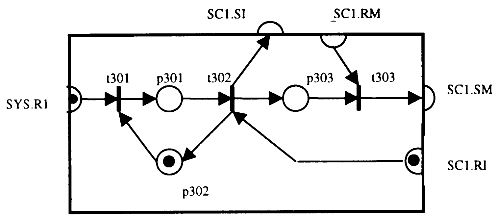

Fig. 16. Behavior model for verifying cc_5.

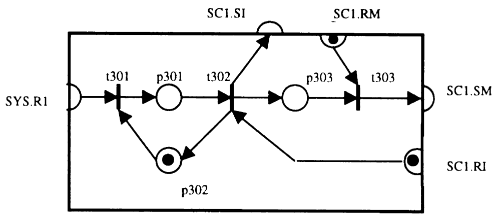

Fig. 17. Behavior model for verifying cc_8.

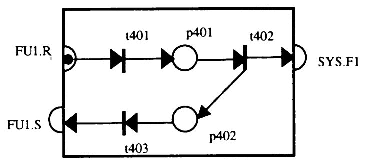

Fig. 18. Behavior model for verifying cc_{11} andcc_{14}.

4. Verifying C2 system

The goal of verification is to show that system design satisfies requirements. The modular nature of SAM 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 constraint-driven refinement and verification method for SAM architectural models. Across design levels (interface and constraint conforming), sub-architectures can be built and analyzed incrementally; at a given design level, a system model can be constructed and analyzed compositionally. Behavioral analysis is driven by constraints verification and carried out through the reachability analysis on TPN [9] and constraint-driven reductions. There are three noteworthy features of our reachability analysis, which help control the complexity of system verification:

1. The reachability analysis in SAM is compositional such that each element (component or connector) in an architecture or sub-architecture can be analyzed individually. Each analyzed element is then reduced to a constant-sized Petri net based on the constraints imposed on the element. The architecture is consequently viewed as a composition of these small nets, and analyzed accordingly.

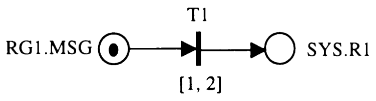

Fig. 19. Behavior model of connector 1.

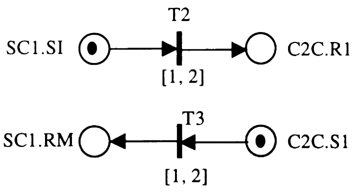

Fig. 20. Behavior model of connector 4.

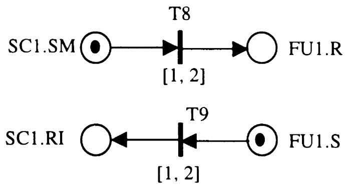

Fig. 21. Behavior model of connector 7.

2. The analysis is incremental across different design levels such that it is performed separately according to the different abstraction levels.
3. The analysis is driven by satisfaction of architectural constraints, which is monitored during the construction of reachability tree, and terminated as soon as the goal is reached. It avoids the generation of a complete reachability tree and thus improves the efficiency.

The verification technique works on the following three steps:

1. verifying module (component/connector) constraints;
2. verifying environment constraints with regard to each individual module;
3. verifying composition level constraints involving multiple modules through incremental structure reduction guided by proven constraints.

Later we show how the algorithm works on the top-level design of the C2 example.

4.1. Verifying module (component/connector) constraints (specifications)

To verify a module, we assign an initial marking to its Petri net behavior model. Figs. 14–18 show the initialized behavioral models for component constraints of C2C, RG1, SC1, and FU1. The difference between these models and those shown in Figs. 4–7 is that here initial markings are set according to the constraints. For example as constraint cs_1 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 in Fig. 14, we add a token to each of the input ports C2C.Ri, $i = 1, 2, 3$.

We first consider the verification of component C2C. The initial marking M_0 is as shown in Table 9. Define marking M_e as in Table 10. Applying simple reachability analysis [9] gives the time delay interval that the model takes from marking M_0 to M_e, denoted by $D(M_0, M_e)$, is [9,13], which implies that interface specification cc_1 is proven satisfied.

Using the same technique, constraints cc_5 and cc_8 can be proven based on Figs. 16 and 17, respectively, and

Table 9

	C2C.R1	C2C.R2	C2C.R3	p201	p202	p203	p204	C2C.S1	C2C.S2	C2C.S3
M_0:	1	1	1	0	0	0	0	0	0	0

Table 10

	C2C.R1	C2C.R2	C2C.R3	p201	p202	p203	p204	C2C.S1	C2C.S2	C2C.S3
M_e:	0	0	0	0	0	0	0	1	1	1

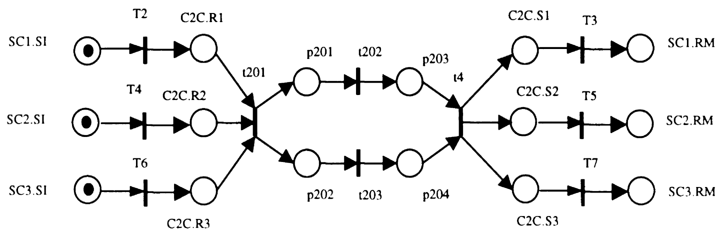

Fig. 22. Behavior model for constraint ec_4.

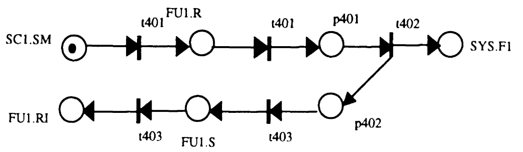

Fig. 23. Behavior model for constraint ec_1.

constraints cc_{11} and cc_{14} can be proven based on Fig. 18. Similarly, constraints cc_6, cc_7, cc_9, cc_{10}, cc_{12}, cc_{13}, cc_{15}, and cc_{16} can be proven satisfied.

Now we consider the constraint cc_2. Reachability analysis shows that the TPN model shown in Fig. 15 has infinite markings as shown in Table 11. However the reachability set has the following properties: (1) places $p_{101} \backsim p_{107}$ are safe; (2) as long as each of transitions t_{101}, t_{102} and t_{103} fires one time, a token is guaranteed to be deposited in port RG1.MSG; and (3) at any moment when each of transitions t_{101}, t_{102} and t_{103} has fired n ($n \geq 1$) times, n or $n - 1$ tokens must have been deposited in port RG1.MSG. Based on these properties, it is easy to see that the net exhibits the same behavior from marking (0 0 0 0 0 0 0 i) to marking (0 0 0 0 0 0 0 $i+1$) as from marking (0 0 0 0 0 0 0 j) to marking (0 0 0 0 0 0 0 $j+1$), $i \geq 1, j \geq 1, i \neq j$. We can easily derive that the time for marking (0 0 0 0 0 0 1) to be reached from the initial marking (1 1 0 0 0 0) is [34,36]. So the component constraints cc_2 is proven. Owing to the symmetry, constraints cc_3 and cc_4 are proven satisfied.

Connector models are usually very simple and easy to verify. Figs. 19–21 show the initialized behavior models for connectors 1, 4 and 7, respectively. These connector constraints, hence all connector constraints due to the symmetry can be proven satisfied straightforwardly.

4.2. Verifying environment constraints

There are four environment constraints to verify. Figs. 22 and 23 show the initialized behavior models for constraints ec_4 and ec_1. Using the same technique as in verifying module constraints, we can easily prove these two constraints, hence all environment constraints due to the symmetry.

4.3. Verifying composition constraints

As a composition constraint is defined on several components, the verification model for a composition constraint is a composition of several components. To control the

Table 11

	$p101$	$p102$	$p103$	$p104$	$p105$	$p106$	$p107$	RG1.MSG	
M_{4i}:	1	1	1	0	0	0	0	i	
M_{4i+1}:	0	0	0	1	1	1	0	i	$i = 0, 1, 2,\ldots$
M_{4i+2}:	0	0	0	0	0	0	1	i	
M_{4i+3}:	0	0	0	0	0	0	0	$i+1$	

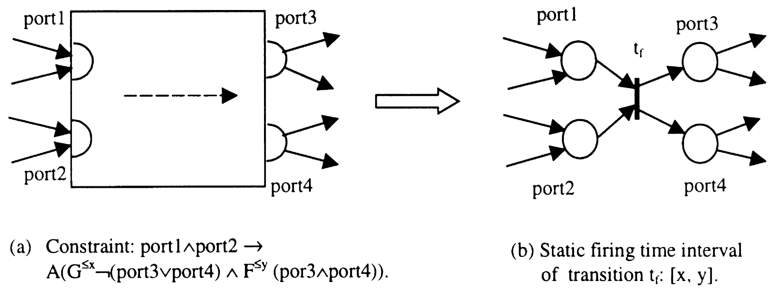
port1
port3
port2
port4
port1
port3
t_f
port2
port4
(a) Constraint: port1∧port2 → A(G^{≤x}¬(port3∨port4) ∧ F^{≤y} (por3∧port4)).
(b) Static firing time interval of transition t_f: [x, y].

Fig. 24. Illustration of component reduction—Case 1.

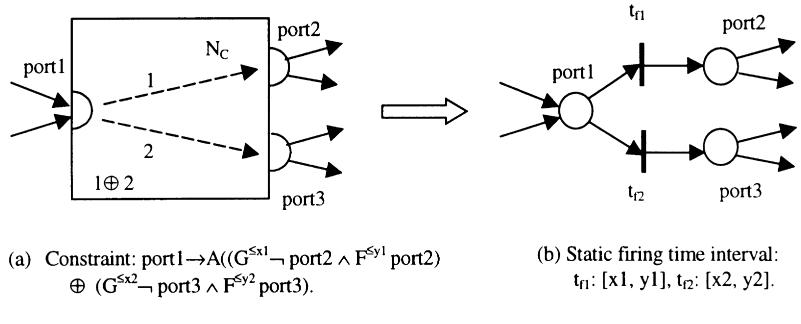
port1
N_C
port2
1
2
1⊕2
port3
t_f1
port2
port1
t_f2
port3
(a) Constraint: port1→A((G^{≤x1}¬ port2 ∧ F^{≤y1} port2) ⊕ (G^{≤x2}¬ port3 ∧ F^{≤y2} port3).
(b) Static firing time interval: t_f1: [x1, y1], t_f2: [x2, y2].

Fig. 25. Illustration of component reduction—Case 2.

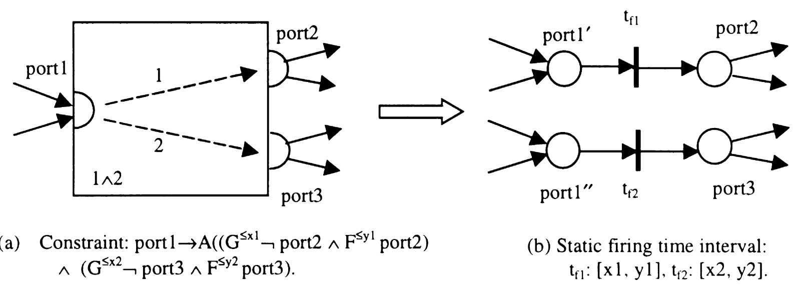
port1
port2
port3
1
2
1∧2
(a) Constraint: port1→A((G^{≤x1}¬ port2 ∧ F^{≤y1} port2) ∧ (G^{≤x2}¬ port3 ∧ F^{≤y2} port3).
port1′
port1″
t_{f1}
t_{f2}
(b) Static firing time interval: t_{f1}: [x1, y1], t_{f2}: [x2, y2].

Fig. 26. Illustration of component reduction—Case 3.

complexity of verification, we are going to reduce a component that appears in the verification model for some composition constraint to a TPN with very small size. The basic idea of the reduction is to use one or more new transitions to connect input and output ports directly, while delete the internal part of the component. The reduction is guided by component constraints. We may have two classes of component constraints. The first class of component constraints indicates the timing requirement for message transfer between a component's input ports and output ports, such as cc_1, cc_5–cc_{16}. The second class of component constraints deals with the requirements for the unconditional output properties of a component, such as cc_2–cc_4. Figs. 24–27 show four typical cases of components reduction based on their component constraints, where the first three cases are based on the first class of component constraints, and the last case is based on the second class of component specifications.

Now we consider the verification of the only compose specification of the C2 example, pc_1. Except for the three radar groups, all components are to be involved to verify the compose constraint. To reduce the complexity, based on their component constraints, we simplify the behavior models of components C2C, SC1 and FU1 to those as shown in Fig. 28(a)–(c), respectively. Components SC2, SC3, FU2 and FU3 can be similarly simplified due to the symmetry among all the sub-center and fire unit modules. We then obtain a simple behavior model for the verification of compose constraint pc_1 as shown in Fig. 29. By reachability analysis, we can obtain that the time for marking at which places SC1.SM, SC2.SM and SC3.SM are marked to be reached from the initial marking is [2, 34], which implies that constraint pc_1 is proven satisfied.

4.4. Violation of constraints

The aforementioned C2 system example illustrated the case that the system design satisfies constraints. If, during verification, a timing constraint is found being violated by the model of architectural design, we need to modify design model. In other words, redesign the system by modifying the component model(s) that violated 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. As the SAM model tightly integrates architectural constraints with every component of the system at every design level, it is easier to make such modifications.

For example, suppose that the component constraint cs_1 is:

$$\text{C2C.R1} \wedge \text{C2C.R2} \wedge \text{C2C.R3}$$

$$\rightarrow AF^{\leq 12}(\text{C2C.S1} \wedge \text{C2C.S2} \wedge \text{C2C.S3}).$$

It indicates that the whole processing time of the component C2C is required to be less than or equal to 12 time units instead of 22 time units as given previously. Then we might find that the component C2C shown in Fig. 5 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*204 to [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%.

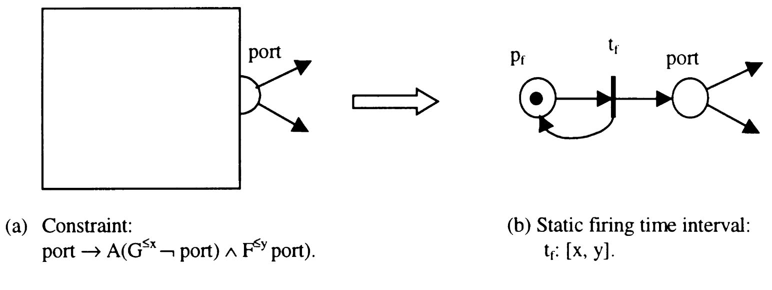
port
(a) Constraint: port → A(G^{≤x} ¬ port) ∧ F^{≤y} port).
p_f
t_f
port
(b) Static firing time interval: t_f: [x, y].

Fig. 27. Illustration of component reduction—Case 4.

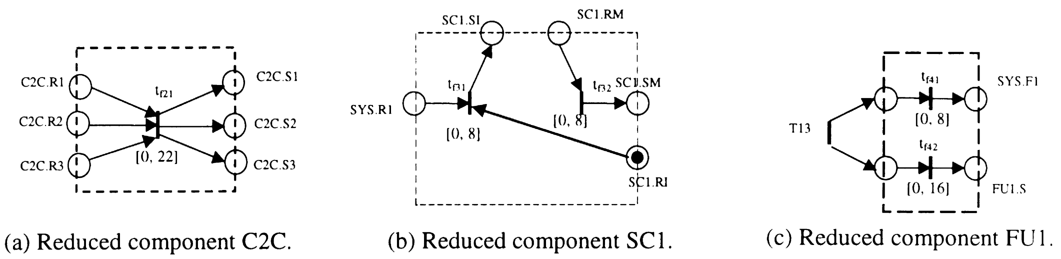
(a) Reduced component C2C. (b) Reduced component SC1. (c) Reduced component FU1.

Fig. 28. Reduction of component models.

5. Summary

In this article, we presented a software architecture SAM through a C2 system. Our objectives of this article are to introduce the main features of SAM without discussing the details of the underlying formal foundation and methodological issues (that are to be published elsewhere) and to provide a useful example.

Although SAM has shared similar philosophies and goals as several other software architecture approaches such as formal notation, executability, and modularity; SAM has many distinct and unique features, including:

1. A well-defined unified formal model integrating two well known complimentary formal notations (temporal logic and Petri nets),
2. A constraint-driven design approach to maintain consistency and completeness horizontally (among modules in the same composition) and vertically (between modules at the different abstraction levels),
3. A compositional and reductive verification technique.

SAM is not an architecture description language such as Wright or Rapide [2] and thus does not provide any specific syntactical sugar in defining interfaces. SAM puts more emphasis on the process of specifying and verifying software architectures rather than the product (the representation) of software architectures. As a result, SAM offers the following benefits:

1. Flexibility—we can easily extend or change the underlying formal specification notations while maintaining the SAM overall framework. For example, we used a high-level Petri net model called predicate transition nets in specifying several well-known architectural connectors recently [25]. By using first order temporal logic and high-level Petri nets, a SAM component/connector specification (S, B) can express almost all interface constituent types (with the exception of the *private* constituent) in Rapide: *public*—input ports (input places in B), *extern*—output ports (output places in B), *constraint*—property specification *S*, and *behavior*—Petri net model B.
2. Analyzability—existing analysis techniques from the underlying formal methods can be directly used to analyze SAM software architecture specifications.

SAM is under constant evolution. Currently, we are working on several directions to improve and extend SAM. Firstly, we will use high-level Petri nets and first-order temporal logic as our formal foundation to enhance SAM's expressive power. For example, it will enable us to handle the specification and verification of requirements that require multiple input tokens in a component. Our earlier work on integrating high-level Petri nets and temporal logic [15] establishes a good foundation to achieve this goal. Secondly, we will develop more heuristics and guidelines to apply SAM to develop architecture specifications. As far as we know, there are very few existing results dealing with a general approach to develop software architecture specifications. Thirdly, we will formalize and automate our current compositional and reductive verification technique. When we adopt high level Petri nets and first order temporal logic, we will further explore and adapt formal proof techniques such as the work by Abadi and Lamport [19–21]. Fourthly, we will explore the possibility of direct code generation from an architecture specification. We will adapt our results on generating CC++ and Java code skeletons from Petri nets [24,26]. Fifthly, we are examining the applicability of SAM in terms of other system properties such as security and dynamic configurability. Lastly, we are constructing a software environment to support the use of SAM.

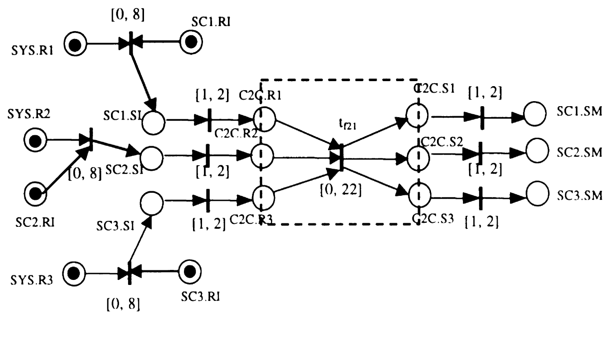
Fig. 29. Reduction of the composition model.

Acknowledgements

We thank the two anonymous referees for careful reading of an earlier version of this article and for making helpful suggestions.

Appendix A. A brief introduction to underlying formal methods of SAM

In this appendix, we give a brief introduction to TPN and RTCTL, which are two underlying formal methods of our SAM model.

A.1. Time Petri nets

A TPN is a tuple (*P*, *T*, *I*, *O*, M_0, *SI*) where:

1. *P* is a finite nonempty set of places;
2. *T* is a finite nonempty, set of transitions; it will appear in the sequel that it may be convenient to view it as an orders set;
3. *I* is the backward incidence function, where *N* is the set of nonnegative integers;
4. *O* is the forward Incidence function;
5. M_0 is the Initial Marking function, (*P*, *T*, *I*, *O* and M_0 together define a Petri net);
6. *SI* is a mapping called static interval, $SI : T \longrightarrow Q^* \times (Q^* \cup \infty)$, where Q^* is the set of positive real numbers.

Let $SI(t_1) = (\alpha^s, \beta^s)$ for some transition t_i, then the interval of numbers (α_i^s, β_i^s) is called the static firing interval of transition t_i the left bound α_i^s the static earliest firing time (state EFT for short), and the right bound β_i^s the static latest firing time (static LFT for short).

A state *S* of a TPN is a pair $S = (M, I)$ consisting of a marking *M* and a firing interval set *I* which is a vector of possible firing times. The number of entries in this vector is given in the number of the transitions enabled by marking *M*.

Transition t_i is *firable* from state $S = (M, I)$ at time $\tau + \theta$ iff:

1. t_i is enabled by marking *M* at time τ, i.e. $(\forall p)(M(p) \geq I(t_i, p))$;
2. The relative firing time θ to the absolute enabling time τ, is not smaller than the FET of transition t_i and not greater than the smallest of the LFT's of all the transitions enabled by marking *M*, i.e. EFT of $t_i \leq \theta \leq$ min {LFT of t_k}, where *k* ranges over the set of transitions enabled by *M*.

Assume that transition t_i be firable at time $\tau + \theta$ from state $S = (M, I)$. Then the state $S' = (M', I')$ reached from *S* by firing t_i at the relative time θ can be computed as follows:

1. M' is computed, for all places *p*, as $(\forall p)M'(p) = M(p) - I(t_i, p) + O(t_i, p)$;
2. I' is computed in three steps:
   - Remove from the expression of *I* the intervals that are related to the transitions disabled when t_i is fired.
   - Shift of the value θ towards the origin of times all remaining firing intervals, i.e. the intervals that remain enabled and so remain in *I*, and truncate them, when necessary, to nonnegative values.
   - Introduce in the domain the static intervals of the new transitions enabled.

A.2. Real-time computational tree logic

An RTCTL Formula is defined as:

1. Each atomic proposition *P* is a formula.
2. If *p*, *q* are formulae, then so are $p \wedge q$ and $\neg\ p$.
3. If *p*, *q* are formulae, then so are *A*(*p U q*), *E*(*p U q*), and *EX p*.
4. (4) If *p*, *q* are formulae and *k* is any natural number, then so are $A(pU^{\leq k}q)$and $E(pU^{\leq k}q)$.

A formula of RTCTL is interpreted with respect to a temporal structure $\Sigma = (S, R, L)$, where *S* is a set of states, *R* is a binary relation on *S* that is total (so each state has at least one successor), and *L* is a labeling which assigns to each state a set of atomic propositions, those intended to be true at the state. Intuitively, this temporal structure Σ represents the reachability graph of the architecture. A full-path $x = s_0, s_1, s_2, \ldots$in Σ is an infinite sequence of states such that $(s_i, s_{i+1}) \in R$ for each *i*; intuitively, a full-path captures the notion of an execution sequence.

References

[1] M. Shaw, D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice-Hall, Englewood Cliffs, NJ, 1996.
[2] D.C. Luckham, J. Kenney, L. Augustin, et al., Specification and analysis of system architecture using Rapide, IEEE Trans. Software Engng 21 (4) (1995) 336–355.
[3] P. Inverardi, A. Wolf, Formal specification and analysis of software architectures using the chemical abstract machine model, IEEE Trans. Software Engng 21 (4) (1995) 373–386.
[4] Y. Deng, J. Wang., R. Sinha, Incremental architectural modeling and verification of real time concurrent systems, Proceedings of the Second IEEE International Conference on Formal Engineering Methods, Brisbane, Australia, 1998.
[5] J. Wang, Y. Deng, Incremental modeling and verification of flexible manufacturing systems, Int. J. Intelligent Manufacturing (in press).
[6] C. Heitmeyer, D. Mandrioli, Formal methods for real-time computing: An overview, Formal Methods for Real-time Computing, 1995, pp. 1–29.
[7] G. Bucci, E. Member, Compositional validation of time-critical systems using communicating time Petri nets, IEEE Trans. Software Eng. 21 (12) (1995) 969–992.
[8] Y. Deng, W. Du, P.C. Attie, M. Evangelist, A formal approach for architectural modeling and decomposition of distributed real-time systems, Proceedings of the Eight International Conference on Software Engineering and Knowledge Engineering, Nevada, 1996, pp. 408–417.
[9] B. Berthomieu, M. Diaz, Modeling and verification of time dependent systems using time Petri nets, IEEE Trans. Software Engng 17 (3) (1991) 259–273.
[10] T. Murata, Petri nets properties, analysis and applications , Proc. IEEE 77 (4) (1989) 541–580.

[11] J. Wang, Timed Petri Nets: Theory and Application, Kluwer Academic Publishers, Dordrecht, 1998.
[12] J.M. Wing, A specifier's introduction to formal methods, IEEE Computer 23 (9) (1990) 8–24.
[13] E.M. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Trans. Programming Languages and Systems 8 (2) (1986) 244–263.
[14] E.A. Emerson, A.K. Mok, A.P. Sistla, J. Srinivasian, Quantitative temporal reasoning, Real-Time Systems 4 (1992) 331–352.
[15] X. He, J.A.N. Lee, Integrating predicate transition nets and first order temporal logic in the specification of concurrent systems, Formal Aspects of Computing 2 (3) (1990) 226–246.
[16] M. Felder, D. Mandrioli, A. Morzenti, Proving properties of real-time systems through logical specifications and Petri net models, IEEE Trans. Software Engng 20 (2) (1994) 127–141.
[17] D. Mandrioli, A. Morzenti, M. Pezze, P.S. Pietro, S. Silva, Formal Methods for Real-time Computing A Petri net and logic approach to the specification and verification of real time systems, Wiley, New York, 1996.
[18] E. Clarke, J. Wing, Formal methods: State of the art and future, ACM Comput. Surv. 28 (4) (1996) 626–643.
[19] M. Abadi, L. Lamport, The existence of refinement mappings, Theoretical Computer Sci. 82 (2) (1991) 253–284.
[20] M. Abadi, L. Lamport, Composing specifications, ACM Trans. Programming Languages and Systems 15 (1) (1993) 73–130.
[21] M. Abadi, L. Lamport, Conjoining specifications, ACM Trans. Programming Languages and Systems 17 (3) (1995) 507–534.
[22] M. Athans, Command and control (C2) theory: a challenge to control science, IEEE Trans. on Automatic Control AC-32 (4) (1987) 286–293.
[23] K. Gerber, D. Kang, S. Hong, M. Saksena, Formal Methods for Real-time Computing A process algebra method for the specification and analysis of real-time system, Wiley, New York, 1996.
[24] X. He, W. Yao, Translating hierarchical predicate transition nets into CC + + program skeletons, Proceedings of the 21st International Computer Software and Application Conference (COMPSAC'97), Washington, DC, 1997, pp. 64–69.
[25] X. He, F. Zeng, Y. Deng, Specifying software architectural connectors in SAM, Proceedings of 11th International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, 1999 (submitted).
[26] S. Lewandowski, X. He, A Java framework for implementing hierarchical predicate transition nets, Proceedings of 10th International Conference on Software Engineering and Knowledge Engineering, San Francisco, CA, 1998.