An Approach for Modeling and Analysis of Security System Architectures

Yi Deng, *Member*, *IEEE*, Jiacun Wang, *Senior Member*, *IEEE*,
Jeffrey J.P. Tsai, *Fellow*, *IEEE*, and Konstantin Beznosov, *Member*, *IEEE*

Abstract—Security system architecture governs the composition of components in security systems and interactions between them. It plays a central role in the design of software security systems that ensure secure access to distributed resources in networked environment. In particular, the composition of the systems must consistently assure security policies that it is supposed to enforce. However, there is currently no rigorous and systematic way to predict and assure such critical properties in security system design. In this paper, a systematic approach is introduced to address the problem. We present a methodology for modeling security system architecture and for verifying whether required security constraints are assured by the composition of the components. We introduce the concept of security constraint patterns, which formally specify the generic form of security policies that all implementations of the system architecture must enforce. The analysis of the architecture is driven by the propagation of the global security constraints onto the components in an incremental process. We show that our methodology is both flexible and scalable. It is argued that such a methodology not only ensures the integrity of critical early design decisions, but also provides a framework to guide correct implementations of the design. We demonstrate the methodology through a case study in which we model and analyze the architecture of the Resource Access Decision (RAD) Facility, an OMG standard for application-level authorization service.

Index Terms—Software security, security system architecture, access control, authorization service, formal architectural modeling, constraint patterns, formal verification, Petri nets, temporal logic.

1 Introduction

Software systems today are increasingly interconnected and accessed in networked environment. This trend is greatly accelerated by rapid proliferation of the Internet. As such, software security has emerged as a foremost concern for modern information enterprise. How to design highly dependable security systems that ensure secure access to distributed software and information is an urgent problem.

Given the magnitude and complexity of distributed systems and information resources interconnected by the Internet and/or enterprise networks, the design of security systems that protect the systems and resources also becomes an increasingly complex and difficult problem. Access control, for example, must consistently and reliably enforce organization-wide security policies across different applications. Security mechanisms must be efficient enough to be useful. An attractive security system design must effectively support system evolution, such as changes in security policies, user population, and their roles, and changes in applications. Furthermore, software security needs to be achieved at reasonable cost during the development, operation, and evolution of the systems.

Security system architecture, which defines the structure of the system, the interaction and coordination among its components, plays a key role in security system design to address the above challenges. Increasingly, security mechanisms are designed as self-contained components or subsystems outside applications in heterogeneous distributed environment [1], [3], [5], [18], [21], [25], [43]. The separation of security logic from application logic in design simplifies the development of both distributed systems and their security functions and, therefore, makes it easier to enhance their quality. Equally important, it paves the way for uniformly applying security mechanisms across (heterogeneous) system boundaries, as well as for centralizing security administration and management in an organization, a traditionally time consuming, costly, and error prone process. Several well-known security system architectures and models, including those in CORBA [5], [31], EJB [19], DCE [18], and DCOM, are cornerstones for designing scalable and flexible security systems in distributed environment. Application level security system models, e.g., those of [3], [15] [21], [22], [33], [43], [44], are expected to gain increasing acceptance.

Despite the advances, however, how to analyze the design of security systems to ensure its consistency and integrity is still a largely open problem. In particular, the composition of security systems is not only to make constituent components work together, but also to ensure that the components as a whole behave consistently and guarantee certain end-to-end properties. A critical property, for example, is whether the system consistently assures

- *Y. Deng is with the School of Computer Science, Florida International University, Miami, FL 33199. E-mail: deng@cs.fiu.edu.*
- *J. Wang is with Nortel Networks at Richardson, Richardson, TX 75082. E-mail: jiacwang@nortelnetworks.com.*
- *J.J.P. Tsai is with the Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053. E-mail: tsai@eecs.uic.edu.*
- *K. Beznosov is with Concept Five Technologies, 25 Bulington Mall Rd., Bulington, MA 07803. E-mail: beznosov@concept5.com.*

Manuscript received 12 June 2000; revised 12 Feb. 2001; accepted 14 Feb. 2001.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 112260.

organization-wide security policies that it is supposed to enforce.

Currently, there is no rigorous and systematic way in the literature to predict and assure critical properties in architectural composition of security systems. In particular, there is no formal way to describe security system architecture, no precise and systematic means to specify required properties that the architectural composition of the security system must satisfy, and no technique to predict and verify that the composition of the system satisfies the properties. Although formal verification of security protocols has received increasing attention in recent years [4], [6], [8], [9], [24], [26], [27], [37], these techniques are generally based on abstract computation models and are not concerned with composition or architecture of security systems. Many of these formal models or techniques are developed for a single security model and do not scale well.

To address the problems, we introduce in this paper a systematic and formal methodology to model security systems architectures and to verify whether required security constraints are assured by the composition of the components of the systems. We argue that such an approach not only helps to ensure the integrity of early design decisions, but also provides a framework to guide correct implementations of the security system design. The result presented in this paper is, to the best of our knowledge, among the first efforts on systematic composition and analysis of security system architectures in the literature.

Our methodology consists of several aspects: First, we present a structured and flexible way for describing security system architectures using the Software Architecture Model (SAM) [40]. Second, we introduce the concept of security constraint patterns, which provides a generic form to formally specify security policies that the security system must enforce. We present a technique to decompose system-wide constraint patterns onto individual components of the system based on the security architecture model and to verify the consistency between global and component constraints. These constraint patterns define what conditions or properties each component and their composition must satisfy under the system architecture. Because the constraints are specified as generic patterns, their usefulness is not limited by a specific set of security policies. Third, in concert with the architecture model and constraint propagation, we present a flexible and scalable technique to verify whether the security system architecture satisfies the required security constraints. Last, but not least, we integrate the above aspects into a systematic and incremental process of security system architecture modeling and verification.

Our methodology follows the following broad steps:

1. We model the security system as a composition of subsystems or components without considering internal details of the components.
2. Security constraints are formalized into system-wide generic constraint pattern(s) which define the security function that the system as a whole must enforce.
3. Based on the security architecture model, the global constraints are decomposed into component constraint patterns that each component must satisfy. By combining a component constraint pattern with the interface (ports) of the component (defined in the architectural model), we can easily generate a simple component model, which preserves the properties defined by the component constraints. These generated component models are often constant in size.
4. These small component models are then plugged into the overall security architecture. This resultant architectural model is verified against the system-wide security constraint patterns using standard analysis techniques, e.g., reachability analysis. This verification shows the consistency between global and component constraint patterns under the security system architecture (Step 1). And,
5. Once the consistency between system-wide and component constraints is verified, these component constraint patterns serve as the basis for component design. In particular, a more detailed architectural or behavior model for each component can be constructed and verified against the corresponding constraint pattern using the same process described above.

This integrated methodology of modeling and verification focuses on the architectural composition of security systems and is highly flexible and scalable. It is flexible because any component which could be a composition of other components can be safely replaced with an alternative design without reanalysis of the overall system architecture so long as the replacement has the same interface and satisfies the component constraint pattern. This feature is especially useful when we apply different security policies or models to the same security system architecture. It is scalable because it allows us to analyze overall architectural composition without the interference of internal details of component design. Verification is done separately at architectural and component levels. This significantly reduces the complexity. There is no need to compose the results of analysis (once the consistency between system-wide and component constraint patterns is verified). The architectural model and constraint patterns are independent of a specific security model or policies. Therefore, our methodology is general and can be applied to a range of security systems. The modeling and verification are driven by propagation of security constraints in a refinement process that incrementally ensures the consistency and integrity of security architecture.

The underlying notations used in this paper include Petri nets [17], [30] and temporal logic [10]. The former is a well-known operational model well suited for modeling the control and composition of distributed systems. By contrast, the latter, a popular descriptive formalism, is best suited for describing rules and constraints. These two notations are seamlessly integrated [23], [40] in our methodology. More details about the notations used are provided in Section 2 and in Appendix B.

We will demonstrate our methodology through a case study in which we model and analyze the architecture of the Resource Access Decision (RAD) Facility, a standard for application-level authorization service adopted by the Object Management Group (OMG).

The rest of the paper is organized as follows: Section 2 will give a more detailed description of our modeling and analysis framework, notation, and process. An overview of the RAD architecture is provided in Section 3. In Section 4, we present a detailed case study of modeling and analysis of the RAD security architecture. Finally, we conclude the paper in Section 5.

2 Framework for Security System Architectures

In this section, we overview the technique used in this paper to model distributed security system architecture and the constraint-driven process for enforcing and verifying security constraints in the composition of the architecture. Additional details about the modeling notations and the process are further elaborated in Section 4.

2.1 Modeling Technique

The modeling technique and notation used in the paper is based on the Software Architecture Model (SAM) [39], [40]. An overview of SAM is given below. The formal notation of SAM is summarized in Appendix B.

Software architecture has emerged as one of most active subject of R&D from both academia and industry for a good reason. Having a sound architecture has profound impact on reliability, scalability, extensibility, and interoperability, among other quality attributes, of software systems during their lifecycle. A formal methodology to support architecture level design is both necessary and desirable for two reasons: First, as the high-level design abstraction, software architecture proceeds and, logically and structurally, influences other system products. Ensuring good design, preventing and detecting errors in architectural descriptions are fundamental to the quality and cost of the systems. Second, because of its high-level abstraction, software architecture description is less complex compared to a detailed design. Thus, application of formal methods is more likely to succeed.

SAM provides a multiple leveled model and notation for describing different aspects of architecture level design such as structure, behavior, and constraints [12], [23], [40]. Its specification model can be characterized from several dimensions:

1. Structurally, software architecture is specified as multilayered compositions of components and connectors, which can be refined and analyzed individually.
2. The construction and refinement of the architectural model are driven by system requirements (specified as architectural constraints) and their propagation. At each design level, SAM specifies not only the (operational) composition of system components, but also the (descriptive) constraints that the components and their composition must satisfy. Refinement goes in lockstep with the propagation of the constraints. SAM provides certain integrity rules (e.g., structural integrity, constraint consistency, and refinement consistency) to assure design consistency. During architectural design, every decision is traceable backward to the requirements and, conversely, every requirement is traceable forward to architectural decisions and designs. Consequently, design traceability and conformity as defined above is maintained while avoiding ad hoc, accidental design, and unjustified efforts.
3. Notation-wise, SAM is integrative and employs both model-oriented formalism (Petri nets) and property-oriented formalism (temporal logic). In particular, SAM provides two levels of modeling notations. The low (proposition) level SAM model employs (time) Place-Transition nets and (Real-Time) Computational Tree Logic ((RT)CTL) for analyzability and the high level (first-order) SAM model utilizes Predicate/Transition nets (PrT-nets) [17] and First Order Temporal Logic [10], [16] for expressiveness. (The high-level model is used in this paper.) Software architecture is specified by a set-theoretical recursive description, where Petri nets are used to describe components and connectors and temporal logic to specify architectural constraints. These two complementary notations are seamlessly integrated under the SAM framework. We have successfully applied SAM for the modeling and analysis of command and control systems [12], [40] and flexible manufacturing systems [13], [39]. The modeling framework of SAM is illustrated in Fig. 1.

Horizontally, at each design level, a system model can be constructed and analyzed compositionally. Vertically, across design levels, a lower level (interface and constraint conforming) subarchitecture can be built and analyzed incrementally and safely plugged into its parent level architecture without the need for reverifying the entire model. A SAM specification needs to satisfy the following consistentcy constraints:

- 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 (constraints consistency),
- The behavior model for a component or subarchitecture at a given level must satisfy the corresponding architectural constraints imposed on the component or subarchitecture (behavior conformance).
- A subarchitecture at design level $k+1$ must inherit all the ports associated with its corresponding component at level k (Fig. 1) (interface consistency).
- A subarchitecture at design level $k+1$ must conform to all constraints which its corresponding component at level k are subject to (Fig. 1) (vertical consistency).

2.2 Methodology for Security System Architecture Modeling and Analysis

The SAM model does not dictate a specific method of system modeling, refinement, and analysis. However, a well-defined method is essential to guide the process of modeling and analysis. In this section, we introduce such a concrete method for security system architecture modeling and analysis based on the SAM model.

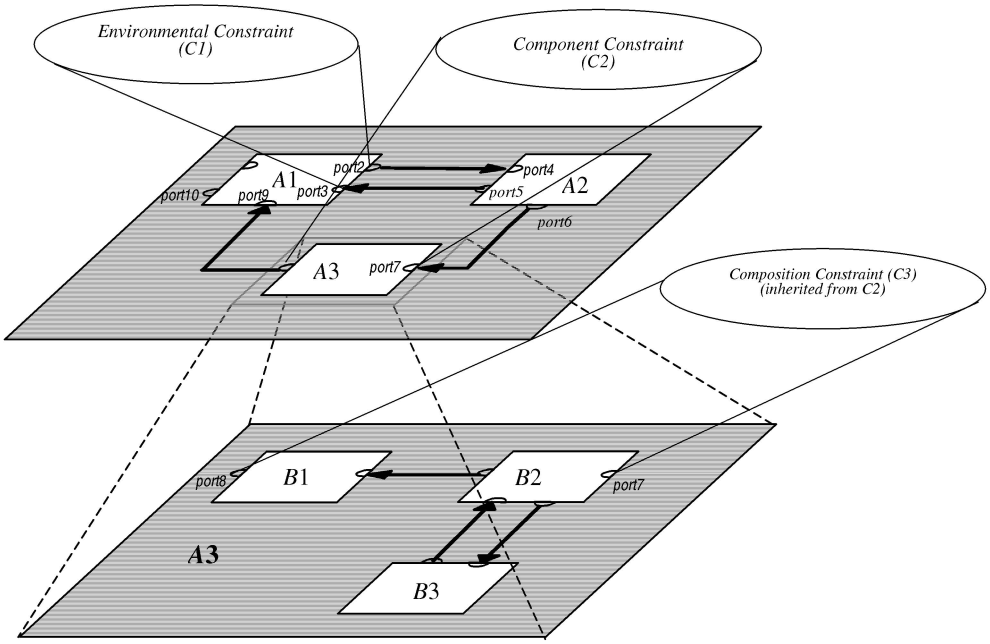


Fig. 1. Modeling framework of SAM.

We introduce the concept of architecture-based security constraint patterns and use propagation of the constraints to drive and guide the composition and verification of security system architecture. The concept of constraints has been widely used in software design and analysis. Definitions, purposes, and applications of constraints vary. For example, system constraints have been defined in the forms of assertions, contracts, pre/postconditions, invariants, etc. [7], [20], [28], [34], [42] to support OO analysis and design. However, what is common is that they represent certain conditions or properties that must be satisfied in system architecturing, design, and implementation. System constraints can also be described or specified in different forms, from informal to formal, ranging from natural languages, to IDL [29], UML, OCL [42], to formal languages and notations, e.g., temporal logic [8], [10], [11], [14] and Petri nets [17], [30]. To enable meaningful, especially automated, reasoning and analysis, however, a certain degree of formalism is necessary. To enable formal verification, rigorously defined mathematical formalism is required. Although a unifying treatment to the concept and application of constraints in software development remains to be seen, their importance is widely agreed.

In our context, we are more interested in how system-wide security constraints are assured in architectural decomposition or refinement. A constraint pattern imposed upon a security system architecture (or a component of the security system) is a generic form of the required security function that the system (or component) must perform and enforce. It is generic in the sense that it is independent of specific security models or policies and can be instantiated when security architecture is applied to the design of a security system based on specific security models or policies. An instance of security constraint may be the specification of security policies of an organization that the security system is set to enforce. (See Section 4.2 for sample policies.) Our concept of constraint pattern has two important implications. 1) Because it is generic, it can be used to constraint the design of a class of security systems rather than a specific one or, in this case, to constraint the composition of a security system architecture which corresponds to a class of systems. 2) The constraint patterns serve as the basis to enforce traceability and consistency in the refinement of the architecture and the basis for verifying whether the composition of the architecture conforms to the security requirement that it supposes to enforce. The focus of this research is not what type or form of security policies should be used in a given context and how to implement them in system design [38] or what types of models and frameworks should be used for composing and designing software systems [29], [32]. Rather, our focuses in this paper are modeling techniques, which can be used to adequately describe security constraints, and methods, which can be systematically applied to reason and/or verify that security constraints are assured in system architecturing and design. For example, one might use the methodology presented in this paper to model and analyze system architectures based the reference models of CORBA [29] or RM-ODP [32]. One such example [3] is given in Section 4. Since our goal is to verify the conformance of security constraints in system architecture, our modeling technique is based on the formal notations of temporal logic and Petri nets.

By incorporating the propagation of the constraint patterns with the architectural refinement process, we achieve an incremental process of verification that is both flexible and scalable. The modeling and verification are driven by the propagation of security constraints in a refinement process that incrementally assures the consistency and integrity of the security system architecture. By introducing a novel technique to ensure the consistency between an architecture level constraint pattern and its corresponding component level constraint patterns, we

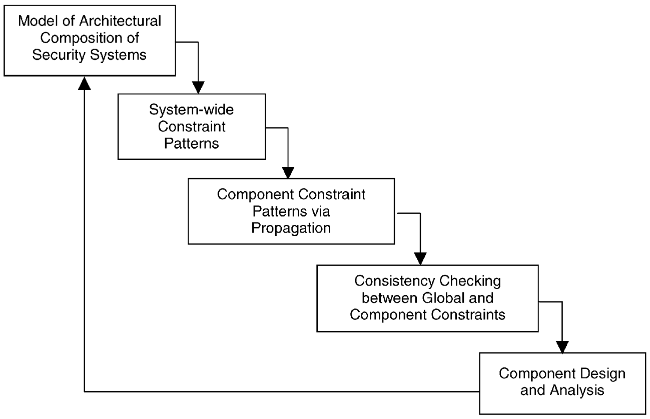


Fig. 2. Process of security system architecture modeling and analysis.

show that the verification of the system architecture can be done separately at architecture and component levels. There is no need to compose the results of analysis at different levels, which can be difficult and costly in conventional compositional analysis. Because of this feature, a component architecture and its model can be easily replaced with alternative designs that conform to the component constraint without the need to reanalyze the overall security architecture.

As shown in Fig. 2, our modeling and analysis process consists of the following major steps.

Step 1. Construction of top-level security system architecture model. The purpose of this step is to build the model for the top-level security system architecture, which describes the overall organization of the system, as well as the coordination and synchronization between its components. Consequently, the internal structure and behavior for the components are not included in this model. This model is constructed by decomposing the system into components and their connections. Component interfaces represented by input ports and output ports are defined, as well as control and data flows between the components.

Step 2. Specify system-wide architectural security constraint patterns. We use first order temporal logic to formally represent the system-wide security constraints imposed on the architectural model created in the previous step. These constraint patterns are specified using only the interface (ports) of the components. The importance of formalizing these original security constraint patterns is twofold: First, the system-wide security requirements are transformed into specific constraint patterns on this particular architecture, more precisely, constraints imposed on the subsystems and connections between the subsystems. Second, by formalizing requirements in terms of architectural constraint patterns, it not only removes ambiguity in the description, but also makes it easier to detect possible inconsistency or conflict between different (competing) requirements.

Step 3. Decompose system-wide security constraint patterns to components. In this step, we decompose the system-wide security constraint patterns to a set of intermediate constraint patterns imposed on the components to guide component design. The constraint pattern defined on a given component specifies the function of that component in terms of its contribution toward the satisfaction of the system-wide constraints under the given architecture. Because the original constraint patterns allow many possibilities for the intermediate constraint patterns, the task of propagating the system-wide constraint patterns on the components requires us to carefully examine and explore the boundary between the components. This is because the propagation of the system-wide constraints to the components effectively partitions the system-wide function to individual component and determines the interface and protocols of interaction between them.

Step 4. Verify consistency between system-wide and component constraint patterns. When the decomposition is done, we need to verify whether the intermediate constraint patterns are consistent with the system-wide security constraint, that is, the component constraints collectively satisfy the system-wide constraints under the given architecture model. Only after these intermediate constraint patterns have been proven to be consistent with the system-wide constraints can it be meaningful to design the components against these intermediate constraint patterns. This verification is facilitated by the facts that 1) the component constraints have similar forms to the system-wide constraints because the former is generated from the latter and 2) the component constraints are connected together by the structure of the architecture model. Therefore, this verification is unlikely to be deterred by the complexity of proving consistency between two arbitrary sets of temporal formulas. A novel verification technique is described in Section 4.

Step 5. Incremental design and verification of the components. The completion of Step 4 has two important implications: 1) The component constraint patterns can be trusted as the basis for component design and 2) if every component design conforms to its component constraints, the resulting system architecture with the inclusion of the component designs will automatically satisfy the system-wide constraint pattern. This is an important conclusion because, as shown in Section 4, it significantly reduces the complexity of analysis. The component design can be used either to construct an operational model of the component conforming to its constraints or to further decompose the component into a subarchitecture. In the first case, component verification can be done using any standard techniques. In the second case, we iterate the above steps, resulting in an incremental architectural composition and analysis process. If necessary, more than one subarchitecture that conforms to the interface and constraints of a component can be developed and plugged into the security system architecture model to evaluate different design alternatives.

3 Overview of the Resource Access Decision (RAD) Facility Architecture

We use the Resource Access Decision (RAD) Facility [3] specification, a standard for application authorization service adopted by the Object Management Group (OMG), as an example to demonstrate our modeling and

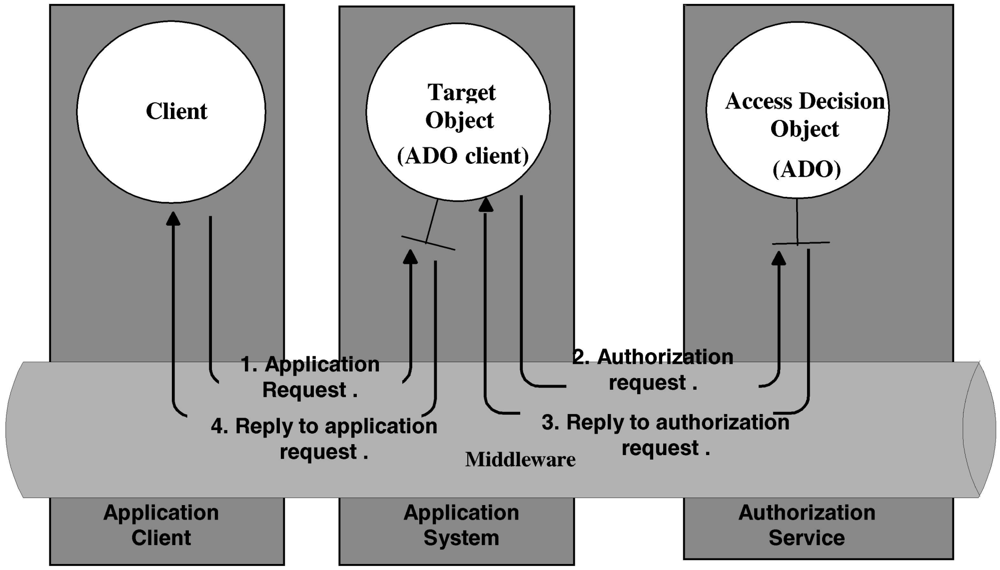


Fig. 3. High-level view of RAD role in authorization decision.

analysis methodology. An overview of the RAD architecture is provided in this section.

The RAD Facility is designed to provide a flexible and general way to handle application-level access control, in particular authorization decisions, in distributed systems. The RAD design is motivated by the fact that the complexity of access control in such application domains as healthcare requires policies that are more sophisticated and of finer granularity than commonly available security mechanisms. The RAD specification provides a security system architecture that encapsulates authorization logic in an authorization service external to the application and is independent of specific security models and policies. Such a security system architecture not only significantly simplifies both application and security system development, but also allows organizations to uniformly manage and enforce their security policies.

To access a protected resource under the RAD architecture, an application requests an authorization decision from a RAD compliant authorization service and enforces the decision. The flow of interactions between application client, application system, and an instance of authorization service is depicted in Fig. 3. The sequence of the interaction is as follows:

1. A client of the application system invokes an operation on the application.
2. While processing the invocation, the application requires an authorization decision from the authorization service.
3. The authorization service makes an authorization decision, which is returned to the application.
4. The application, after receiving an authorization decision, enforces it. If access is granted by the authorization service, the application returns expected results of the invocation. Otherwise, it either returns partial results or raises an exception.

A RAD service is composed of the following components (Fig. 4): The *AccessDecisionObject* (ADO) serves as the interface to RAD clients and coordinates the interactions between other RAD components. Zero or more *Policy-Evaluators* (PEs) perform evaluation decisions based on certain access control policies that govern the access to protected resource. The *DecisionCombinator* (DC) combines the results of the evaluations made by potentially multiple PEs into a final authorization decision by applying certain combination policies. The *PolicyEvaluatorLocator* (PEL), for a given access request to a protected resource, keeps track of and provides references to a DC and, potentially, several PEs which are collectively responsible for making the authorization decision to the request. The *DynamicAttributeService* (DAS) collects and provides dynamic attributes about the client in the context of the intended access operation on the given resource associated with the provided resource name.

Fig. 4 shows interactions among components of authorization service. They are outlined below and readers are referred to [3] for more details about the RAD architecture:

1. The authorization service receives a request via the ADO interface.
2. The ADO obtains object references to those PEs associated with the resource name in question and an object reference for the responsible DC.
3. The ADO obtains dynamic attributes of the principal (client) in the context of the resource name and the intended access operation.
4. The ADO delegates an instance of DC for polling the PEs (selected in Step 2).
5. The DC obtains decisions from PEs and combines them according to its combination policy.
6. The decision is forwarded to the ADO, which in turn returns the decision to the application.

4 Architectural Modeling and Analysis of RAD—A Case Study

In this section, we discuss the modeling and analysis of the Resource Access Decision (RAD) architecture based on the

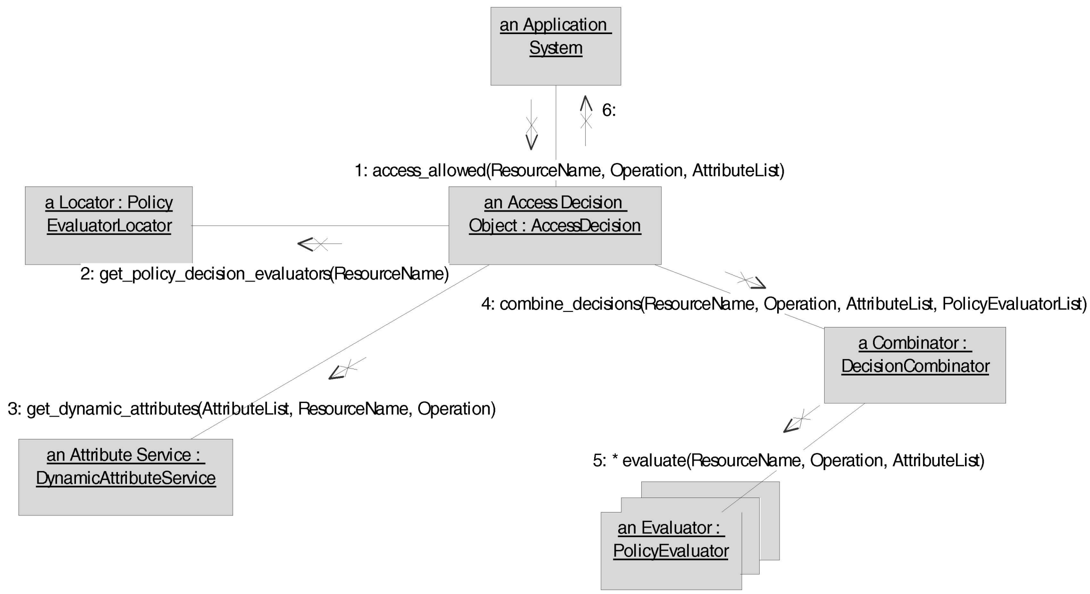


Fig. 4. RAD interaction diagram.

framework and process discussed in Section 2. Our case study begins with building the high-level architecture model of RAD, based on which system-wide security constraint patterns are formulated and specified. Based on the architecture model, we propagate system-wide constraint patterns to the system's constituent components. After the consistency between the system-wide constraint patterns and component constraint patterns is verified, we develop individual component model against its component constraint patterns. We will also show how our modeling and analysis process supports dynamic change of security policies under the RAD architecture.

4.1 High-Level Architecture Model of RAD

The high-level architecture model describes the structural composition of a system, intercomponent connections, component interfaces, and overall control flow. We divide the RAD components shown in Fig. 4 into two groups. The first group includes ADO, PEL, DAS, and the second DC and PEs. The components in the first group are generally independent of specific security policies, while the design of the second group is affected by specific policies and thus more dynamic in this sense. We model ADO, PEL, and DAS as defined in the RAD architecture while considering DC and PEs as one component in the top-level model to be decomposed later. This makes it easier for us to plug in different DC and PE designs in case of different security policies. We also include an application system component in the model, which acts as the environment for the RAD model. The resultant model is shown in Fig. 5 with its variables explained in Table 2. We use Petri nets to represent the interface of components and their connections, where communication ports of the components are represented by places (half circles) on the border of the components. Interactions between the components are modeled by simple Petri nets.

The control flow between the components is guided by the following constraints:

$\Box\,(P0 \rightarrow \Diamond P1)$; (when a user issues an access request, the AS will pass the request to the RAD service).

$\Box\,(P2 \rightarrow \Diamond P5)$; (when ADO is invoked by AS, it will invoke PEL and DAS).

$\Box\,(P7 \rightarrow \Diamond P9)$; (when PEL is invoked, it will return references for DC and PEs).

$\Box\,(P8 \rightarrow \Diamond P10)$; (when DAS is invoked, it will return security attributes).

$\Box\,(P11 \rightarrow \Diamond P12)$; (when DC&PEs is invoked, it will return access control decision).

$\Box\,(P6 \rightarrow \Diamond P3)$. (when ADO gets decision from DC&PEs, it will return the decision to AS).

These constraints, combined with the structural connections between the components (Fig. 5), lead to a basic system-wide property reachability that needs to be guaranteed represented by the following logic formula:

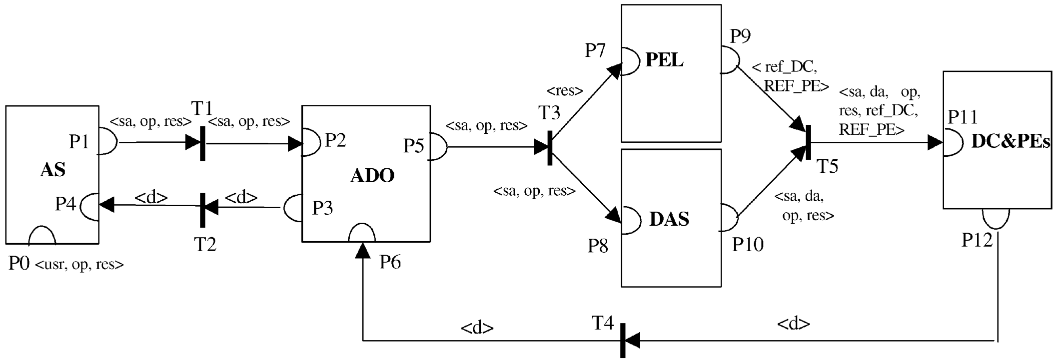
$$\Box\,(P0 \rightarrow \Diamond P4). \tag{1}$$

It says that, when a client requests an access, it is guaranteed to receive a response from the RAD.

Two observations can be made here. First, no internal details about the components are revealed at this step. Second, even though flow relations between the components are specified, the precise boundaries between the components are not defined until constraint patterns associated with the components are defined.

4.2 Architecture Constraint Patterns and Their Consistency Verification

In this section, we first formally specify the system-wide security constraints. We then define intermediate component constraints to guide the design of components. Finally, we verify the consistency between the system-wide constraints and intermediate component constraints. This is an important step to assure the security requirements during system design.



Reachability constraints:

□(P0 → ◇P1), □(P2 → ◇P5), □(P7 →◇P9), □(P8 → ◇P10), □(P11 → ◇P12),

□(P6 → ◇P3), □(P0 → ◇P4)

Fig. 5. Top-level architecture composition of RAD.

4.2.1 Sample Security Policies

For the purpose of illustration, we consider a set of simplified but typical access control policies in the healthcare domain, which arguably has one of most complex security requirements. Consider a hospital information enterprise consisting of many application systems. They are used for registration, billing, collecting results of laboratory tests and transcribed X-ray images, as well as for storing all other clinical information about patients including records of their visits to the hospital (for out-patients) and their stay overnight (for in-patients).

Hospital employees involved in the care process are called caregivers. A caregiver accesses many of those clinical, laboratory, transcription, and financial systems either directly with specialized client software or via general-purpose application programs. Such programs interact with several of application servers in order to provide caregivers with information needed for patient diagnosis and treatment.

Let us assume that the hospital adopts the policies listed in Table 3 to control access to patients' medical records.

4.2.2 System-Wide Security Constraint Patterns

As discussed in Section 2, our approach of security system architecture composition and analysis is driven by satisfaction of system-wide security constraints to assure required access control policies in the composition of the architecture. Since the most important end-to-end property of RAD is the assurance of security policies, the construction of the behavioral model starts with the formulation of a generic form of access control policies, which serves as the pattern of architectural constraints for the RAD design.

The system-wide constraints are defined on ports P0 and P3 (Fig. 5), which, when marked, denote the input request and output decision of the RAD, respectively. Our study shows that any security policy is composed of three entities: 1) a subject, which issues the access request on behalf of a client, 2) a resource name, representing the protected resource, on which an operation is to be performed, and 3) the name of the operation. To formally describe the polices, we define a normal form, op(*sb*, *res*), to represent that subject *sb* can perform operation *op* on the resource *res*. For example, Policies (P1.1) and (P1.2) of Table 3 can be described as:

(R1.1) read (caregiver, patient's name).
(R1.2)
  (R1.2.1) modify (registration clerk, patient's name),
  (R1.2.2) modify (registration clerk, demographic information).

This way, the given policies in Table 1 form a set $PL = \{op_i(sb_i, res_i), i = 1, 2, \ldots, n\}$, where

$$sb_i \in \{"caregiver," "registration clerk," "nurse," "technician," "assistant physician," "physician," "psychiatrist"\}.$$

$$op_i \in \{"read," "modify,"\} \text{ and}$$

$$res_i \in \{"PN," "DD," "CDD," "CRR," "CSR," "CRT," "CST," "PRR," "PSR," "PRT," "PMI"\}.$$

TABLE 1
Legends for Fig. 5

Port (Place)	Description	Type
P0	User resource access request	usr×op×res
P1	User resource access request with static security attribute	sa×op×res
P2	ADO invoked	sa×op×res
P3	Decision to user resource access request	d
P4	Decision to user resource access request received by application	d
P5	Invocation of functions issued at RAD ADO	sa×op×res
P6	Final decision received at RAD ADO	d
P7	PEL invoked	res
P8	DAS invoked	sa×op×res
P9	References of DC and PEs	ref_DC×REF_PE
P10	Dynamic attributes	sa×da×op×res
P11	DC&PEs invoked	sa×da×op×res ×ref_DC×REF_PE
P12	Final decision received by RAD DC	d
Transition	**Description**	
T1	Transmit a user's resource access request to RAD ADO	
T2	Transmit the decision to a user's request to application	
T3	Transmit a user's request to RAD process	
T4	Transmit the decision to RAD ADO	
T5	Invoke DC&PEs	

TABLE 2
Variables in Fig. 5

Variable	Description
usr	User name
sa	Static attributes of the user
da	Dynamical attributes of the user
op	Requested operation
res	Resource name
ref_DC	Object reference of a DC
REF_PE	Object reference set of policy evaluators
d	Access control decision. d ∈ {'Y', 'N'}. 'Y'—granted, 'N'—denied.

TABLE 3
Sample Access Control Polices of a Hospital (Policies 1)

No.	Description
P1.1	Any **caregiver** can read patient's name.
P1.2	**Registration clerk** can modify patient name and demographic information.
P1.3	**Nurse** can read patient's name and demographic information, modify current episode demographic information, can read current episode regular records and current episode regular test results.
P1.4	**Technician** can modify current episode regular and sensitive test results.
P1.5	**Assistant physician**, in addition to what a nurse can do, can also read all regular records of patients.
P1.6	**Physician**, in addition to what assistant physician can do, also can modify current episode regular and sensitive records, and read regular and sensitive records and test results from previous episodes.
P1.7	**Psychiatrist**, in addition to what a physician can do, also can modify mental information.

(See Table 5 in Appendix A for abbreviations of the resource names.)

Also, we denote by $S(usr, res)$ the set of subjects associated with user usr regarding resource res. By generalizing the above discussion, the system-wide security constraint for the RAD architecture can be described as:

$$\begin{aligned}&\forall(usr, op, res, d)\\&\Box\,(P0.(usr, op, res) \wedge (\exists\ sb \in S(usr, res), op(sb, res) \in PL)\\&\qquad\rightarrow \Diamond P3.d \wedge (d = 'Y'))\\&\wedge \Box\,(P0.(usr, op, res) \wedge (\forall sb \in S(usr, res), op(sb, res) \notin PL)\\&\qquad\rightarrow \Diamond P3.d \wedge (d = 'N')).\end{aligned} \tag{2}$$

This constraint specifies that if there exists a subject in set $S(usr, res)$ that is allowed by given security policies to perform operation op on resource res, then the RAD must grant the access request; otherwise, the RAD must deny the access request. Note that this formula is independent of specific policies and provides a general pattern for system-wide security constraints. This pattern can be instantiated because the set PL may represent any group of security policies.

4.2.3 Intermediate Component Constraint Patterns

The assurance of the system-wide security constraint is achieved through the collaboration of the RAD components. To guide component design, we introduce a set of *intermediate constraints* that specify the requirements for the components. These intermediate constraints, or *component constraints*, together can be viewed as products of the decomposition of the system-wide security constraints. Collectively, the component constraints should satisfy the system-wide constraint based on the RAD architecture. We now consider the intermediate constraint patterns for each of the RAD components:

AS (application system). When the AS issues an access request on behalf of a user, the AS is required to provide the static attributes regarding the user (e.g., John is a physician), which is a parameter used in the authorization decision. Let the static attributes of user usr be $sa(usr)$. To enforce the system-wide security policies, we ask that when a user logs onto the security service system, an authentication service provide correct static attributes of the user. That is, the AS is subject to the following pattern of constraints on its interface ($P0$ and $P1$).

$$\begin{aligned}&\Box\,(P0.(usr, op, res) \rightarrow\\&\Diamond P1.(sa, op, res) \wedge (P1.sa = sa(usr))).\end{aligned} \tag{3}$$

Notice that AS is not part of RAD. This pattern is a constraint to the environment of the RAD.

ADO (access decision object). ADO acts as a facade that provides a single and uniform interface to other interfaces, which interact with a RAD service. Its responsibility is to coordinate the execution of other RAD components and pass the final decision to AS. The intermediate constraint patterns for this component are defined as:

$$\begin{aligned}&\Box\,(P2.(sa, op, res) \rightarrow \Diamond P5.(sa, op, res)\\&\wedge (P2.(sa, op, res) = P5.(sa, op, res))),\end{aligned} \tag{4}$$

$$\Box\,(P6.d \rightarrow \Diamond P3.d \wedge (P3.d = P6.d)). \tag{5}$$

Formula (4) requires that, when ADO receives a resource access request (at port $P2$), it should forward the request to PEL and DAS via port $P5$. Formula (5) requires that, when ADO receives the final decision regarding a request, it must forward the decision to the application system via port $P6$.

PEL (policy evaluator locator). When the ADO receives an authorization decision request, it needs to know which DC and PEs are applicable to the given resource. Component PEL shoulders the responsibility. It provides references to a DC and one or more PEs that are needed to perform the evaluation of the access request. In fact, there is a multiple-to-one mapping between resource names and DCs and a multiple-to-multiple mapping between resource names and PEs. We denote by *ref_DC(res)* the object reference of the DC that is responsible for the evaluation of requests regarding resource res and denote by *REF_PE(res)* the set of object references of the PEs that are responsible for the evaluation of requests regarding resource *res*. To enforce the system-wide policies, we ask the PEL return correct object references of DC and PEs. So, the component is subject to the following pattern of constraint:

$$\begin{aligned}\Box\,(P7.res \rightarrow \Diamond P9.(ref\_DC, REF\_PE) \\ \wedge (P9.ref\_DC = ref\_DC(res)) \\ \wedge (P9.REF\_PE = REF\_PE(res))).\end{aligned} \tag{6}$$

It specifies when PEL receives a request (at port $P7$) with a given resource name (res), it must return a DC's object reference and one or more PEs' object references (at port $P9$), where the DC *(ref_DC(res))* and the PEs *(REF_PE(res))* are responsible for the evaluation of requests about the given resource.

DAS (dynamic attributes service). For certain dynamic policies, which are time or context sensitive (e.g., only an attending physician can modify a patient's record), a PE needs to know the "dynamic attributes" of the user or principal (e.g., user John is the a*ttending* physician for patient Mary) with respect to the resource to be accessed. A dynamic attribute is determined at the time an access request takes place. DAS is responsible for acquiring and providing the dynamic attributes for the principal in the context of the intended access operation on the given resource with the provided resource name. We denote by $da(sa, res)$ the dynamic attributes regarding sa and res. To enforce the system-wide security policies, we ask the DAS to return correct dynamic attributes. Hence, we have the following pattern of constraint for DAS:

$$\begin{aligned}\Box\,(P8.(sa, op, res) \rightarrow \Diamond P10.(sa, da, op, res) \\ \wedge (P8.(sa, op, res) = P10.(sa, op, res)) \\ \wedge (P10.da = da(sa, res))).\end{aligned} \tag{7}$$

It indicates when DAS receives a request (at port $P8$) with the static attributes of a user and the name of a resource that the user is going to access, it must return the dynamic attributes of the user regarding the resource (at port $P10$).

DC&PEs (decision combinator and policy evaluators). The compound component DC&PEs is responsible for making an authorization decision for a given resource access request. As long as it receives correct static attributes, dynamic attributes, and object references of DC and PEs, the component must make a decision consistent with the access control policies. That is, it is subject to the following pattern of constraints:

$$\begin{aligned}&\forall (sa, da, op, res, ref\_DC, REF\_PE, d),\\ &\Box\,(P11.(sa, da, op, res, ref\_DC, REF\_PE) \wedge (sa = sa(usr))\\ &\quad \wedge (da = da(usr) \wedge (ref\_DC =\\ &ref\_DC(res) \wedge (REF\_PE = REF\_PE(res))\\ &\quad \wedge (\exists\, sb \in S(sa, da, res), op(sb, res) \in PL)\\ &\quad \rightarrow \Diamond P12.d \wedge (d = 'Y'))\\ &\wedge \Box\,(P11.(sa, da, op, res, ref\_DC, REF\_PE) \wedge (sa = sa(usr))\\ &\quad \wedge (da = da(usr) \wedge (ref\_DC =\\ &ref\_DC(res)) \wedge (REF\_PE = REF\_PE(res))\\ &\quad \wedge (\forall\, sb \in S(sa, da, res), op(sb, res) \notin (PL)\\ &\quad \rightarrow \Diamond P12.d \wedge (d = 'N')),\end{aligned} \tag{8}$$

where

$$\begin{aligned}S(sa, da, res) = S(sa(usr), da(sa), res) = \\ S(sa(usr), da(sa(usr)), res) = S(usr, res).\end{aligned} \tag{9}$$

This constraint pattern specifies that, if component DC&PEs receives the correct static attributes (sa), dynamic attributes (da) of the principal, and set of PEs, the component must make a decision consistent with the security policies. That is, if the intended access operation (op) on the requested resource (res) by the user represented by the principal is allowed by security policies, the component must grant the access request; otherwise, it must deny the request.

Although the above constraint patterns are derived from the system-wide constraint pattern (2), we cannot be certain that the corresponding component constraints ((3)-(8)) are consistent with the system-wide constraints (2) under the RAD architecture without explicit verification. The consistency between these two sets of constraints is the basis to assure the consistency of the composition of the architecture. Only after these intermediate constraints have been proven to be consistent with the system-wide constraint can it be meaningful to design the RAD components against these intermediate constraints. In the next section, we present a novel technique to verify the consistency of these two sets of constraint patterns.

4.2.4 Verifying Consistency between Constraint Patterns

With system-wide and component constraint patterns defined, our next step is to verify the consistency between them. Generally speaking, verifying the consistency between two arbitrary sets of first order temporal formulas is not decidable. However, we have two leverages here. First, the component constraints are "derived" from the system-wide constraints. Therefore, they share similar forms and structures. Second, we have the connectors of the component constraints available. This connector is the architectural model described by Fig. 5, which links the component constraints together. Armed with these two pieces of information, we introduce a technique to check the consistency between the two sets of temporal patterns, which consists of the following steps:

1. Assume C is the set of components in the security system architecture. From each constraint pattern for component $c \in C$, we derive a small and constant-sized PrT-net, which we call the *component requirement model* of c (against the constraint pattern), denoted as CRM(c). CRM(c) can be constructed by translating the temporal formula representing the component constraint into its PrT-net form. Notice that CRM(c) has the same ports as c because the formula is defined on the ports, i.e., these ports constitute the vocabulary of the formula. It is easy to see that CRM(c) describes the required behavior of c against the constraint pattern.
2. We plug the set of newly created Petri nets $\{CRM(c)|c \in C\}$ into the security architecture model (also represented as a Petri net, e.g., Fig. 5), which results in a complete, i.e., executable, net model. Let us call this net the *constraint model of the architecture*, which represents the model of the

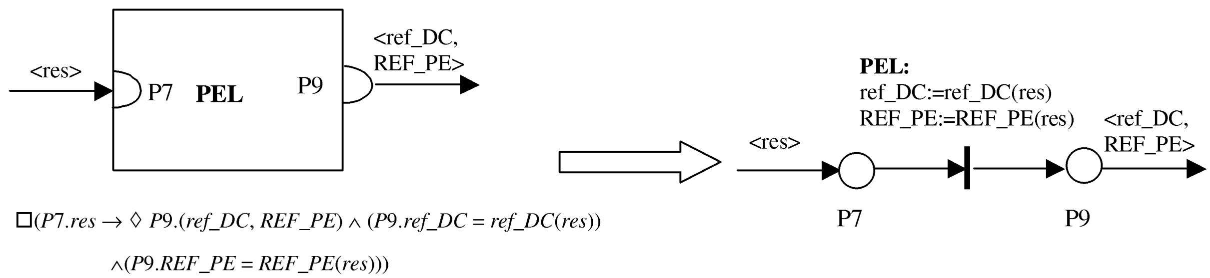


Fig. 6. Illustration of getting the requirement model of component PEL.

component constraint patterns under the system architecture. This implies that if this Petri net satisfies the system-wide constraint patterns, then the component constraint patterns are consistent with the system-wide patterns based on the security system architecture.

3. Verify if the constraint model satisfies the system-wide constraint patterns. A number of available techniques, e.g., reachability analysis, can be used for this verification.

Constructing component requirement models and architecture constraint model of RAD. Creating a CRM is to convert each component in the security architecture, which is currently a black box, to a simple PrT net based on its component constraints. Notice that each component constraint given in Section 4.2.3 specifies the relationship (or mapping) between input token(s) (at the input ports of the components) and the output token(s) (at the output ports). We connect the input and output ports with a transition t and impose that relationship as an assertion to the transition. The inscription on the arc from the input port to t has the same structure as the input token and the inscription on the arc from t to the output port has the same structure as the output token. The resultant component requirement model describes component constraints consistent to the interface of the component.

For example, as shown in Fig. 6, component PEL can be represented by a PrT net that has two places $P7$ and $P9$ and a transition PEL. The assertion imposed on transition PEL is $ref\_DC := ref\_DC(res)$ and $REF\_PE := REF\_PE(res)$, which comes from the constraint pattern for PEL (6). A set of general rules which guide the conversion of a component constraint pattern to a component requirement model are described in [41].

When every RAD component is converted into its requirement model, we get a complete PrT net model of the RAD architecture, as shown in Fig. 7, which describes the system-wide behavior as defined by the component constraint patterns based on the RAD architecture model.

Verifying consistency between system-wide and component constraint patterns. The consistency verification is carried out by analyzing whether the execution of the architecture constraint model shown in Fig. 7 produces conflicting results against the system-wide constraint patterns. Standard reachability analysis technique of PrT nets is used here for the verification [17].

The initial marking is set as such that port $P0$ contains a token with attribute $< usr, op, res >$, while no other places contain token. We denote initial marking by $M0 = P0.(usr, op, res)$ for simplicity.

- The firing of transition AS at $M0$ produces marking $M1 = P1.(sa, op, res)$, where $sa = sa(usr)$,
- The firing of transition $T1$ at $M1$ produces marking $M2 = P2.(sa, op, res)$, where $sa = sa(usr)$,
- The firing of $ADO1$ at $M2$ produces marking $M3 = P5.(sa, op, res)$, where $sa = sa(usr)$,

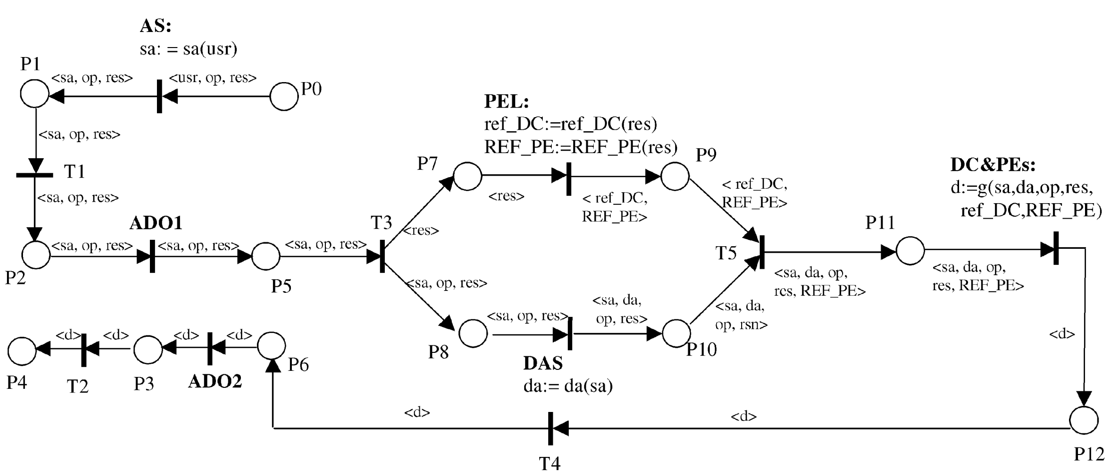


Fig. 7. Architecture constraint model of RAD.

- The firing of $T3$ at $M3$ produces marking
$$M4 = P7.res P8.(sa, op, res),$$
where $sa = sa(usr)$,
- The firing of transition PEL at $M4$ produces marking
$$M5 = P8.(sa, op, res)P9.(ref\_DC, REF\_PE),$$
where
$$sa = sa(usr), ref\_DC = ref\_DC(res),$$
and $REF\_PE = REF\_PE(res)$,
- The firing of transition DAS at $M4$ produces marking
$$M6 = P7.(res)P10.(sa, da, op, res),$$
where $da = da(sa)$ and $sa = sa(usr)$,
- The firing of transition DAS at M5 produces marking $M7 = P9.(ref\_DC, REF\_PE)P10.(sa, da, op, res)$, where
$$sa = sa(usr), da = da(sa), ref\_DC = ref\_DC(res),$$
and $REF\_PE = REF\_PE(res)$,
- The firing of transition PEL at $M6$ also produces marking
$$M7 = P9.(ref\_DC, REF\_PE)P10.(sa, da, op, res),$$
where
$$sa = sa(usr), da = da(sa), ref\_DC = ref\_DC(res),$$
and $REF\_PE = REF\_PE(res)$,
- The firing of transition $T5$ at $M7$ produces marking $M8 = P11.(sa, da, op, res, ref\_DC, REF\_PE)$, where
$$sa = sa(usr), da = da(sa), ref\_DC = ref\_DC(res),$$
and $REF\_PE = REF\_PE(res)$.

The relationship $d = g(sa, da, op, res, REF)$ (defined on transition *DC&PEs*) is determined by the constraint defined on component DC&PEs (8). Since, at $M8$, the relations
$$sa = sa(usr), da = da(sa), ref\_DC = ref\_DC(res),$$
and $REF\_PE = REF\_PE(res)$ are guaranteed for any given (usr, op, res) at $P0$, so we can rewrite (8) as

$$\begin{aligned}&\forall(usr, op, res),\\ &\Box\,(P0.(usr, op, res) \wedge (\exists\, sb \in S(sa, da, res), op(sb, res) \in PL)\\ &\qquad \rightarrow \Diamond P12.d \wedge (d = 'Y'))\\ &\wedge \Box\,(P0.(usr, op, res) \wedge (\forall\, sb \in S(sa, da, res), op(sb, res) \notin PL)\\ &\qquad \rightarrow \Diamond P12.d \wedge (d = N))\end{aligned} \tag{10}$$

Combining (10) and (9) gives

$$\begin{aligned}&\forall(usr, op, res),\\ &\Box\,(P0.(usr, op, res) \wedge (\exists\, sb \in S(usr, res), op(sb, res) \in PL)\\ &\qquad \rightarrow \Diamond P12.d \wedge (d = 'Y'))\\ &\wedge \Box\,(P0.(usr, op, res) \wedge (\forall\, sb(S(usr, res), op(sb, res) \notin PL)\\ &\qquad \rightarrow \Diamond P12.d \wedge (d = 'N')).\end{aligned} \tag{11}$$

Notice that, when transition *DC&PEs* fires and deposits a token with attribute $< d >$ to place $P12$, the firings of transitions $T4$ and $ADO2$ will carry a token with the same attribute $< d >$ to place $P3$. In other words, we always have
$$P3.d = P12.d. \tag{12}$$
It follows from (11) and (12) that

$$\begin{aligned}&\forall(usr, op, res),\\ &\Box\,(P0.(usr, op, res) \wedge (\exists\, sb \in S(usr, res), op(sb, res) \in PL)\\ &\qquad \rightarrow \Diamond P3.d \wedge (d = 'Y'))\\ &\wedge \Box\,(P0.(usr, op, res) \wedge (\forall\, sb \in S(usr, res), op(sb, res) \notin PL)\\ &\qquad \rightarrow \Diamond P3.d \wedge (d = 'N')).\end{aligned}$$

This is exactly the same formula as the original system-wide security constraint of the system (2). Hence, we conclude that the component constraint patterns are consistent with the system-wide security constraint pattern.

4.3 Component Modeling and Verification

The consistency between the system-wide and component constraint patterns is important. This is because, once it is proven, we can proceed with the design of the components and, as long as the behavior of the components satisfies their corresponding constraint patterns, the entire architecture is consistent with system-wide constraints. Notice that the consistency between system-wide and component constraints is based on the given architecture.

Now that we have shown the constraint consistency for the RAD architecture, we are ready to consider component modeling and analysis.

4.3.1 Refinement of Component DC&PEs

The authorization decision for a given resource access request is made by the chosen Policy Evaluators (PEs) and Decision Combinator (DC). There may be several PEs involved in processing a resource access request. The DC collects decisions from each of the PEs and makes a final decision. In the high-level architecture model, the component of DC&PEs is in fact a composition of DC and one or more PEs. For the moment, we assume one PE with role-based access control (RBAC) is used, which is sufficient to support Policies 1 (Section 4.2.1). (Readers are referred to Appendix A for an overview of RBAC and its application on Policies 1). Fig. 8 shows the architecture model of DC&PEs, which supports role-based authorization. In the figure, variable $d1$ indicates the decision made by the RBAC PE.

Obviously, the composition of the DC and PE is subject to the constraint of (8). To guide the design of components DC and PE, we need to further define intermediate component constraint patterns on them. These component

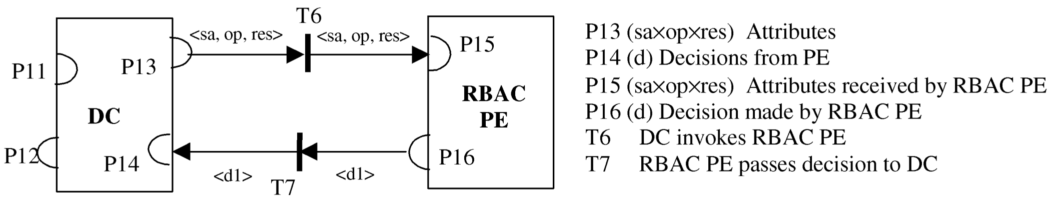


Fig. 8. Composition of DC&PE—One PE Case.

constraint patterns together must be consistent with the composition constraint of (8).

To formulate the constraint pattern for the PE, we represent the permission assignment relation given in Table 7 in Appendix A as a set PA such that if role rl is allowed to perform operation op on a resource named res according to this table, then we have $(rl, op, res) \in PA$. This way, we get

$$\begin{aligned}&PA = \{(\text{Psychiatrist}, \text{M}, \text{AMD}),\\&(\text{Physician}, \text{M}, \text{CRR}), (\text{Physician}, \text{M}, \text{CSR}),\\&(\text{Physician}, \text{R}, \text{CST}), (\text{Physician}, \text{R}, \text{PSR}),\\&(\text{Physician}, \text{R}, \text{PST}), \ldots\ldots, (\text{Care-giver}, \text{R}, \text{PN})\}.\end{aligned}$$

Also, we denote by RES the set of resources names. Formula (13) describes the behavioral constraint on the PE:

$$\begin{aligned}&\forall(sa, op, res, d),\\&\Box\,(P15.(sa, op, res) \wedge (res \notin RES) \rightarrow \Diamond P16.d \wedge (d = {}'U'))\\&\wedge \Box\,(P15.(sa, op, res) \wedge (res \in RES)\\&\qquad \wedge (\exists\, rl \in RL(sa), (rl, op, res) \in PA)\\&\qquad \rightarrow \Diamond P16.d \wedge (d = {}'Y'))\\&\wedge \Box\,(P15.(sa, op, res) \wedge (res(RES)\\&\qquad \wedge (\forall\, rl \in RL(sa), (rl, op, res) \notin PA)\\&\qquad \rightarrow \Diamond P16.d \wedge (d = {}'N')).\end{aligned} \tag{13}$$

It says that if the resource named res is not contained in the permission assignment table, the PE returns "U" (stands for *unknown*) to the DC. If the resource is in the permission table and there is at least one rl in $RL(sa)$ allowed to perform operation op on the resource, the PE returns "Y" (stands for *yes* or granted) to the DC. If no rl in $RL(sa)$ is permitted for operation op, it returns N (stands for *no* or denied) to the DC. Notice that the form of this constraint pattern is bound to the RBAC access control model because the PE is a RBAC PE.

Now, we consider the constraint patterns of DC. When the DC is invoked by ADO, it will first invoke the selected PEs, collect decisions from all the PEs, and then make the authorization decision based on certain combination policy. Suppose that the DC uses the following policy to process the decision of the RBAC PE:

$$d = f1(d1) = \begin{cases} 'Y', \text{if } d1 & = \quad 'Y' \text{ or } 'U';\\ 'N', \text{ if } d1 & = \quad 'N'.\end{cases} \tag{14}$$

Two component constraint patterns are defined for the DC:

$$\begin{aligned}&\Box\,(P11.(sa, op, res, REF) \rightarrow \Diamond P13.(sa, op, res)\\&\wedge (P11.(sa, op, res) = P13.(sa, op, res)))\end{aligned} \tag{15}$$

$$\begin{aligned}&\forall(d1, d),\\&\Box\,(P14.d1 \wedge ((d1 = {}'Y' \vee (d1 = {}'U')) \rightarrow \Diamond P12.d \wedge (d = {}'Y'))\\&\wedge \Box\,(P14.d1 \wedge (d1 = {}'N') \rightarrow \Diamond P12.d \wedge (d = {}'N')).\end{aligned} \tag{16}$$

Formula (15) dictates the flow of access request from port $P11$ to $P13$, and (16) specifies that the DC follows (14).

Once the above constraint patterns are formalized, the same technique described in Section 4.2.4 can be used to verify the consistency between the constraint of (8) and intermediate component constraint patterns of (13), (15), and (16).

4.3.2 Model of RBAC Policy Evaluator (PE)

Fig. 9 shows the behavior model of the RBAC PE: When it is invoked by DC ($P15$ is marked) with a message from DC that includes the static attributes (sa) of the principal, the type of operation (op), and the resource name (res), the PE searches the permission assignment table (Table 7 of Appendix A) for the resource named res (transition sr fires). If it doesn't find the resource (transition nfr fires), it returns "U" (stands for unknown) to DC. If it does (transition fr fires), for each role in $RL(sa)$, it checks the permission table to see if operation op on the resource is permitted. If no role is permitted to perform the operation (transition nfp fires), it returns "N" to DC. Otherwise (transition fp fires), it returns "Y" to DC.

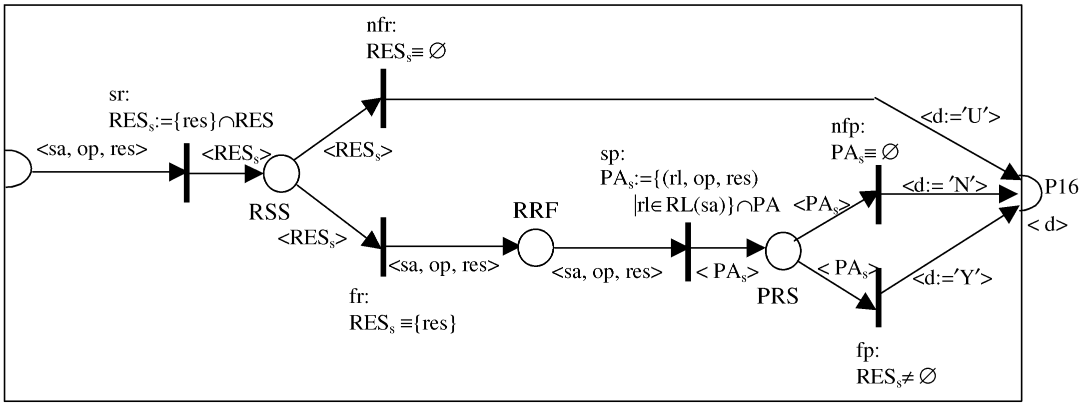
4.3.3 Component Verification

Once we have the behavior model of a component, a number of available techniques, e.g., reachability analysis [30], model checking [11], [14], simulation, theorem proving [37], can be used to verify or check its conformance to the component constraint patterns. Because the techniques are readily available in the literature, we will not describe the verification process here. It is sufficient to say that the component model of Fig. 9 satisfies its constraint pattern of (13).

Upon verifying that all the components satisfy their corresponding constraint patterns, we can conclude that the composition of the security system architecture satisfies its system-wide security constraint pattern because we have proven that those component constraint patterns are consistent with the system-wide security constraint patterns.

4.4 Extended RAD Model to Support More Complex Policies

In this section, we show that our methodology provides a framework to support changes in security architecture. In



Notes:

RPS (RES): Related resource search result

RRF (sa×op×res): Result of the policy search

PRS (PA): Related policy search result

sr Search in the policy base the named resource in the request

nfr not find the named resource

fr find the named resource

sp Search in the policy base related policies to the request

nfp not find related polices

fr find related polices

Fig. 9. Behavior model of the RBAC PE.

particular, we discuss how our approach helps minimize the impact caused by changes of access control policies in the modeling and analysis of security system architecture.

Policies 1 of Table 3 allows an employee to have certain access to the records of all patients, regardless of whether the employee is involved in the process of providing care to a patient. Let us assume that new legislation requires the hospital in our example to ensure that patient records are accessed according not only to employee functions but also to the fact that the employee is actually involved in the care process for the patient. For example, only the attending physician is now allowed to modify current episode records of the patient. Also, let us assume that now patient relatives, guardians, and designated representatives have the right to limited access to the patient's record. The new set of policies is described below.

4.4.1 New Access Control Policies

The hospital, in order to become compliant with the new legislation, augments its access control authorization policies and replaces Policies 1 with the new policies listed in Table 4.

The new policies require that only caregivers, those participating the treatment process for a given patient, can have access to the patient records according to their job description. This is an example of the *least-privilege* security principle (i.e., minimum privileges needed to complete a task should be granted to a user). However, authorization decisions for such policies can be made only if the (context and time dependent) relationship between the patient and the user (principal) are taken into account. It is very challenging to make such authorization decisions if only RBAC mechanisms are employed, which would require additional control to be exercised via manual procedures in medical records department of the hospital. This prevents complete computerization of medical records and the treatment processes. To avoid this situation, relationships between users and patients whose records are about to be accessed should be computed each time an authorization decision is to be made.

To enforce the new policies, a new PE with relationship-supporting role-based access control (RelBAC) mechanism is needed. The introduction of the new PE causes the change of the structure of the RAD service. In the next section, we show the change is minimized to the reconstruction of compound component DC&PEs.

4.4.2 RAD Reconfiguration to Support Policy Changes

Fig. 10 shows the composition of DC&PEs in the presence of both RBAC PE and RelBAC PE. Refer to Appendix A for the access control model designed for the new policies on which both the (modified) RBAC PE and the (new) RelBAC PE are based. When the DC is invoked by the ADO (P11 marked), it will invoke the two PEs (T6), and then collect their decisions (T7). The final decision of the DC will go to the ADO through port P12. Notice that, although the structure of DC&PEs with one PE is different from that with two PEs, they share the same external interface (ports P11 and P12). This allows both compositions to conform to the base architecture model (Fig. 5) from the structure point of view.

Now, the RBAC PE and RelBACE work together to enforce the new access control policies. However, it is possible to assign each policy to a specific PE based on its

TABLE 4
New Policies (Policies 2)

No.	Description
P2.1	Any **care-giver** can read patient's name.
P2.2	**Registration clerk** can modify patient name and demographic information.
P2.3	**Nurse** can read patient's name and demographic information.
P2.4	**Attending nurse**, in addition to the rights of any other nurse, can modify current episode demographic information, can read current episode regular records and current episode regular test results.
P2.5	**Technician** can read patient's name and modify current episode regular test results.
P2.6	**Related technician**, in addition to the rights of any other technician, can modify current episode sensitive test results.
P2.7	**Attending assistant physician**, in addition to what a nurse can do, can also read all (i.e. from the current and previous episodes) regular records and all regular test results, as well as to modify current episode regular records.
P2.8	**Attending physician**, in addition to the rights of attending assistant physician, can modify current episode sensitive regular records and can read all regular and sensitive records from previous episodes.
P2.9	**Attending psychiatrist**, in addition to what an attending physician can do, also can modify mental information.
P2.10	**Patient relative** can read patient's current episode demographic and patient's name.
P2.11	**Patient guardian** can read previous episode regular data.
P2.12	**Patient spouse** can read previous episode sensitive data.
P2.13	**Patient representative** can read previous episode regular data provided that patient gives a consent.

distinguishing function. By checking the new policies listed in Table 4, we can find that policies P2.1, P2.2, P2.3, and P2.5 are suitable to be evaluated by the RBAC PE, while all other policies are suitable to be evaluated by the RelBAC PE.

5 Discussions and Conclusions

We have presented a formal methodology for the modeling and analysis of software security system architectures. Through the case study of the RAD architecture, we have shown that our methodology provides a systematic way to assure critical security constraints, in particular the enforcement of required security policies, in the architectural composition of security systems or services. We have introduced security constraint patterns in the context of system architecture model as a general and precise way to define critical system properties that must be satisfied in individual system design. We have also presented a technique for consistent propagation of system constraints in architectural refinement to guide the modeling and verification process. It is also shown that the methodology is both flexible and scalable. Verification is done separately at architecture and component levels, which significantly reduces the complexity of analysis.

Our contributions are twofold: a general methodology for assuring security constraints in architectural decomposition and concrete modeling and analysis techniques that provide an implementation to the methodology. We believe that this approach can be applied to different application domains and provide a systematic way to

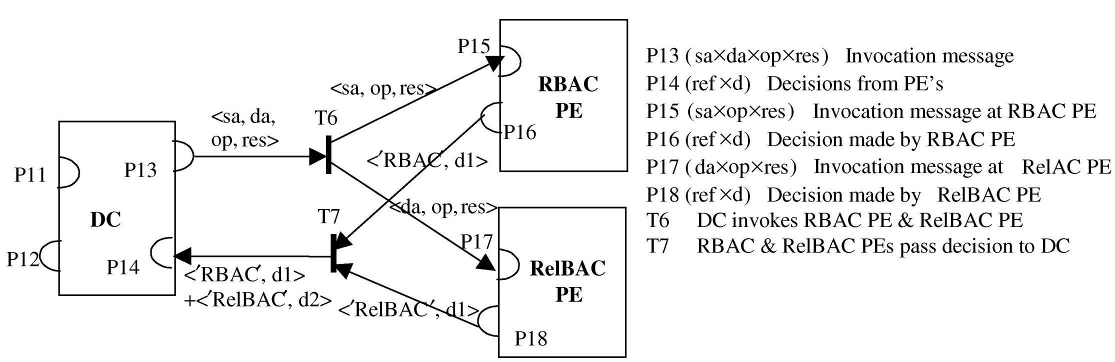


Fig. 10. DC&PEs model for relationship-based authorization.

ensure the compliance of security constraints in design. However, how to assign security constraints to individual components or subsystems is not only a domain specific but also a system specific design issue, which is not and should not be dictated by the methodology presented in this paper. This is because a given assignment represents a specific partition of responsibility or functionality of the components and determines (in part) the interfaces and protocols between the components. Our methodology, however, can be used to ensure that a given partition satisfies the system-wide constraints and the analysis can be used to compare and evaluate different design alternatives. In database systems, for example, though query optimizer does not explicitly handle any security function, its behavior nonetheless affects system security. One might impose security constraints on the module such that the query optimizer cannot change any security attributes associated with the query and it can only accept request from trusted sources. In [38], an elaborate set of security constraints is presented for multilevel secure database management systems (MLS/DBMS) and a distributed system architecture is described to handle security constraint processing in distributed MLS/DBMS environment. It would be an interesting research issue to apply the methodology presented in this paper to verify if indeed the proposed constraint processing architecture of [38] guarantees the security constraints.

The result presented in this paper helps to bridge the gap between the practical design of software security systems and formal analysis that exists today. We believe that such a methodology not only helps ensure the integrity of critical early design decisions of software security systems, but also provides a framework to guide security system implementation. Furthermore, we believe that, in addition to security constraints, the approach presented in this paper is applicable to other critical properties, e.g., performance, availability and fault tolerance, of security system design as well. How to model these critical properties or quality attributes as distinct aspects of the design that coexist under a common architecture framework and how to assess these properties in the architectural composition of security systems, based on the approach introduced in this paper, are currently under investigation.

Appendix A

An Introduction to Role-Based and Relationship-Based Access Control

In RBAC [35], [36], roles are used to describe individuals' function in the organization. The roles are treated as an attribute of an individual. Appropriate permissions are associated with each role for resource access. The role assignment effectively enables the permissions in RBAC mechanisms.

Assume a hospital uses RBAC to control access to patients' records, which are composed of the parts as shown in Table 5. User to role assignment relation (UA) is shown in Table 6 and role hierarchy is shown in Fig. 11 along with permision assignment relation (PA) in Table 7. For simplicity, we have only seven users, from a to g. Each of them, except user d, is assigned to only one role according to their functions in the hospital. The role hierarchy indicates that a user can act in any role junior to the one he or she is assigned in UA. For example, user d can activate any of the following roles: *caregiver*, *technician*, and *nurse* because he/she is assigned roles *nurse* and *technician*. Suppose user e requests to read *sensitive test results from previous episodes* (PST) of a patient. From Table 6 we know that user e is a "Registration Clerk" and, from Fig. 11, a "Registration Clerk" has all rights of a "Care-giver." By checking Table 7, we know that a "Registration Clerk" can only modify PN and DD, and a "Care-giver" can only read "PN." We can conclude that user e is not authorized to read PST.

TABLE 5
Sample Structure of Patient Medical Record

Part name	**Abbreviation**
Patient name	PN
Patient demographic data	DD
Patient current episode demographic data	CDD
Patient current episode regular records	CRR
Patient current episode sensitive records	CSR
Patient current episode regular test results	CRT
Patient current episode sensitive test results	CST
Patient regular records from previous episodes	PRR
Patient sensitive records from previous episodes	PSR
Patient regular test results from previous episodes	PRT
Patient sensitive test results from previous episodes	PST
Patient mental information from all episodes	PMI

A.1 Supporting Relationships in RBAC (Relationship-Based Access Control (RelBAC)

RelBAC uses relationships between entities to support more complex, e.g., context-sensitive, policies [2]. For example, the new policies in Table 4 require considering not only a user's role in the hospital, but also his (time-dependent) relationship to the patients. To achieve this, the role hierarchy is extended into a relationship hierarchy (Fig. 12), which is in turn used to extend the permission assignment relationship by combining role-based assignment (Table 8) with relationship-based assignment (Table 9). Suppose that user b, a physician, requests to read *sensitive records from previous episodes* (PSR) of a patient, and further suppose b is an *attending* physician of this patient. By checking Table 9, we know that an attending physician is permitted to read his patient's PSR. Hence, the request is granted.

Appendix B

Formal Notations of SAM

The underlying formal notation of SAM are *predicate-transition* nets (PrT nets) [17] and *first order temporal logic* [10], [16]. Their notations are summarized below.

B.1 Predicate/Transition Nets

A predicate/transition net consists of the following elements:

1. A directed graph (P, T, I, O), where

TABLE 6
User to Role Assignment Relation

		Users						
		a	b	c	d	e	f	g
Roles	Psychiatrist	✓						
	Physician		✓					
	Physician Assistant			✓				
	Nurse				✓			
	Registration Clerk					✓		
	Technician				✓		✓	
	Care-giver							✓

- P is a finite set of places,
- T is a finite set of transitions, $P \cup T \neq \oslash$, and $P \cap T = \oslash$,
- $I : P \times T \rightarrow N$ is an input function that defines directed arcs from places to transitions, where N is a nonnegative integer,
- $O : P \times T \rightarrow N$ is an output function that defines arcs from transitions to places.

2. A structure set Σ consisting of some types of individual tokens together with some operations and relations.
3. A labeling of arcs with types of token variable (including the zero-attributes indicating a nonargument token). Each label can be a multiple set expression of the form $< k1 \times 1, \ldots, k_n x_n >$, where $\{x_i\}$ is a set of sorted variables, and $\{k_i\}$ a set of constants.
4. An inscription on some transition being a logical formula constructed from the operation and relations of the structure Σ and variables occurring at the surrounding arcs.

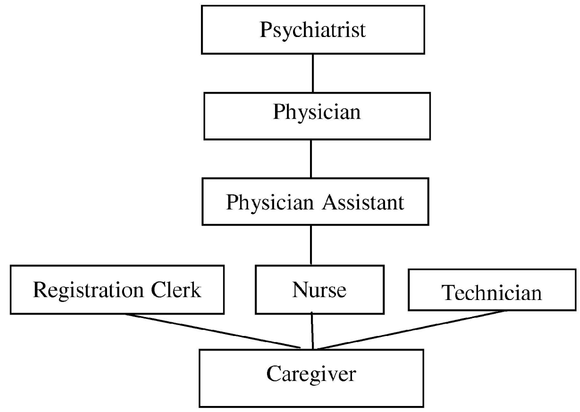


Fig. 11. Role hierarchy.

5. A marking of the places of P with n attributes of individual tokens.

Firing rules: An instance of typed label variables of tokens is an *occurrence mode* of a PrT net. We use $e : \alpha$ to denote the result of instantiating an expression e with α. A transition is α*_enabled* at marking M if $M(p) \geq I(p,t) : \alpha$ for $p \in P$. If t is α*_enabled* at M, t may *fire* in occurrence mode α. The firing of t with α returns the marking M' defined by $M'(p) = M(p)(I(p,t) : \alpha \cup O(t,p) : \alpha$ for $p \in P$. The *state space* of the system consists of the set of all markings connected to the initial marking through such occurrence of firing.

B.2 First Order Temporal Logic

First order temporal logic is a kind of modal logic. The first order temporal logic used in this paper contains the following temporal operators: *always*, represented by $\Box$, *sometimes*, represented by $\Diamond$, and next, represented by O.

As a first order language, first order temporal logic consists of two kinds of expressions: *temporal terms* and *temporal formulas*. Temporal terms are constructed from individual constants, individual variables, function symbols, and temporal operator O. Temporal formulas are constructed from predicate symbols (including equality

TABLE 7
Permission Assignment (PA) Relation for Role Hierarchy (for Policies 1)

		Resources											
		PN	DD	CDD	CRR	CSR	CRT	CST	PRR	PSR	PRT	PST	PMI
Roles	Psychiatrist												M
	Physician				M	M		R		R		R	
	Physician Assistant								R		R		
	Nurse		R	M	R		R						
	Registration Clerk	M	M										
	Technician						M	M					
	Care-giver	R											

TABLE 8
Permission Assignment Relation (PA) for Role Hierarchy (for Policies 2)

		Resources											
		PN	DD	CDD	CRR	CSR	CRT	CST	PRR	PSR	PRT	PST	PMI
Roles	Psychiatrist												
	Physician												
	Physician Assistant												
	Nurse		R	M									
	Registration Clerk	M	M										
	Technician						M						
	Caregiver	R											

and propositions), function symbols, individual constants, individual variables, the classical operators (¬, ∧, ∨, ⊃ , ∈ , and ≡) and quantifiers (∃ and ∀), and the temporal operators (□ , ◇, and O). In this paper, all individual constants, individual variables, and predicate symbols are rigid, function symbols can be rigid or flexible—flexible symbols may change their values in different states, while rigid symbols do not.

Temporal logic formulas are evaluated under computation models. A computation model for temporal logic is a tuple (I, α, and σ), where I is an interpretation that specifies the domain under consideration and assigns meanings to constants, function symbols, and predicate symbols according to their sorts, α is an assignment that assigns a value from the domain of the nonnegative rational numbers with 0 to each of the individual variables, and $\sigma = s_0 s_1 \ldots$ is an infinite sequence of states. In this paper, the set of variables is the set P of places and the set of states is the set of all reachable markings.

B.3 The Software Architectural Model (SAM)

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

TABLE 9
Permission Assignment Relation for the Relationship Hierarchy (Fig. 12)

		Resources											
		PN	DD	CDD	CRR	CSR	CRT	CST	PRR	PSR	PRT	PST	PMI
Relationships	Attending Psychiatrist												M
	Attending Physician					M		R		R		R	
	Attending Physician Assistant				M		R		R		R		
	Attending Nurse			M	R		R						
	Related Technician							M					
	Related Care-giver	R											
	Patient												
	Spouse									R		R	
	Guardian								R		R		
	Relative	R	R										

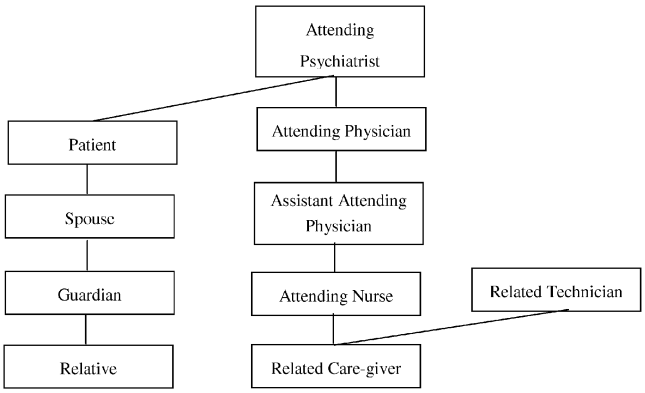


Fig. 12. Relationship hierarchy.

SAM $= (C, h)$, where

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

- Cm is a set of components. For each

$$Cm_j \in C_m, Cm_j = (Sm_j, Bm_j),$$

where

- Sm_j is a property specification (component constraints) of component Cm_j. It is defined by a set of first-order temporal logic formulas.
- Bm_j is the behavior model of component Cm_j. It is defined by a PrT net. Let

$$Cm_j.PORT = \{p | p \in Cm_j.P \wedge \bullet\, p \cap Cm_j.T = \emptyset \wedge p \bullet \cap Cm_j.T = \emptyset\}.$$

$Cm_j.PORT$ is the set of ports of component Cm_j which defines the interface of the component (any place p with an empty input preset, $\bullet\, p \cap Cm_j.T = \emptyset$, is an input port, and any place p with an empty postset, $\bullet\, p \cap Cm_j.T = \emptyset$, is an output port). In addition, there is no common node between any two components, i.e., for

$$\forall Cm_j, Cm_k \in C_i.Cm,$$

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

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

$$c.PORT \subseteq Cm_l.PORT.$$

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

- Cn is a set of connectors. For each

$$Cn_j \in C_n, Cn_j = (Sn_j, Bn_j),$$

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 first-order temporal logic formulas.
- Bn_j is the behavior model of connector Cn_j. It is defined by a PrT net such that

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

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

The above 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:

$$c.PORT \subseteq Cn_l.PORT.$$

The overall behavior (PrT) 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

$$C_i.PORT\_EXT = \{p | p \in \bigcup_{cm_j \in Cm} Cm_j.PORT \wedge \bullet p \cap C_i.T = \emptyset \vee p \bullet \cap C_i.T = \emptyset\}.$$

$C_i.PORT\_EXT$ 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.

- Cs is a set of architectural constraints. Each $Cs_j \in Cs$ is a first-order temporal logic 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 iff:

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

$$\bigwedge_{c \in Sm_i \cup Sn_i \cup Cs} \mathcal{C} \tag{17}$$

2.

$$\forall C_i \in C, \forall Cm_l \in C_i.Cm, h : Cm_l \rightarrow C_j, j \neq i$$

such that

- $$Cm_l.PORT = C_j.PORT\_EXT, \tag{18}$$

- $$Cm_l.Sm_l \subseteq C_j.Cs. \tag{19}$$

Equation (17) states that all constraints should be consistent with each other and it establishes the (*horizontal*) *constraint/specification consistency condition*. Equation (18) states that when refining a component into a subarchitecture, the subarchitecture must inherit all ports of the component as all its external ports, and it establishes the *structural consistency condition*. Equation (19) states that when refining a component into a subarchitecture, the subarchitecture 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. Equations (18) and (19) together give the *vertical* (*interface*) *consistency conditions*.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation under grant No. CCR-0098120 and by the US Army Research Office under grant No. DAAG55-98-1-0428. The views and conclusions contained herin are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied by the above agencies.

REFERENCES

[1] J. Barkley, "Implementing Role-Based Access Control Using Object Technology," *Proc. First ACM Workshop Role-Based Access Control,* pp. 93-98, 1995.
[2] J. Barkley and K. Beznosov et al., "Supporting Relationships in Access Control Using Role Based Access Control," *Proc. ACM Role-Based Access Control Workshop,* pp. 55-65, 1999.
[3] K. Beznosov and Y. Deng et al. "A Resource Access Decision Service for CORBA-Based Distributed Systems," *Proc. Ann. Computer Security Applications Conf.,* pp. 310-319, 1999.
[4] P. Bieber and N. Boulahia-Cuppens, "Formal Verification of Authentication Protocols," *Proc. BCS-FACS Sixth Refinement Workshop,* 1994.
[5] B. Blakley, *CORBA Security: An Introduction to Safe Computing with Objects.* Addison-Wesley, 1999.
[6] D. Bolignano, "Towards a Mechanization Cryptographic Protocol Verification," *Lecture Notes in Computer Science,* vol. 1254, p. 131, 1997.
[7] G. Booch, *Object-Oriented Analysis and Design with Applications,* second ed. Benjamin/Cummings, 1994.
[8] R. Boyer and J. Moore, *A Computational Logic.* New York: Academic Press, 1979.
[9] M. Burrows et al., "A Logic of Authentication," *ACM Trans. Computer Systems* vol. 8, no. 1, pp. 18-36, 1990.
[10] L.S. Cauman, *First-Order Logic.* Berlin: Walter de Gruyter, 1998.
[11] E.M. Clarke et al., "Model Checking and Abstraction," *ACM Trans. Programming Languages and Systems,* vol. 16, no. 5, pp. 1512-1542, 1994.
[12] Y. Deng and J. Wang, "Integrated Architectural Modeling and Analysis for High-Assurance Command and Control System Design," *Annals Software Eng.,* vol. 7, pp. 47-70, 1999.
[13] Y. Deng and C.R. Yang, "Architecturedriven Modeling of Real-Time Concurrent Systems with Application in FMS," *J. Systems and Software,* vol. 45, pp. 61-78, 1999.
[14] E.A. Emerson and A.P. Sistla, "Symmetry and Model Checking," *Proc. Fifth Int'l Conf. Computer Aided Verification,* 1993.
[15] R. Filman and T. Linden, "SafeBots: A Paradigm for Software Security Controls," *Proc. New Security Paradigms Workshop,* pp. 45-51, 1996.
[16] M. Fitting, "First-Order Modal Tableaux," *J. Automated Reasoning,* vol. 4, pp. 191-213, 1988.
[17] H.J.G. Genrich, "High-Level Nets Fundamentals," *Proc. Advances in Petri Nets 1990,* pp. 207-247, 1990.
[18] F. Gittler and A.C. Hopkins, "The DCE Security Service," *Hewlett-Packard J.,* vol. 46, no. 6, pp. 41-48, 1995.
[19] L.M. Gong et al. "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2.," *Proc. USENIX Symp. Internet Technologies and Systems,* pp. 103-112, 1997.
[20] I. Graham, *Migrating to Object Technology.* Addison-Wesley, 1995.
[21] B. Hailpern and H. Ossher, "Extending Objects to Support Multiple Interfaces and Access Control," *IEEE Trans. Software Eng.,* vol. 16, no. 11, pp. 1247-1257, Nov. 1990.
[22] J. Hale et al., "Security Policy Coordination for Heterogeneous Information Systems," *Proc. Ann. Computer Security Applications Conf.,* pp. 219-228, 1999.
[23] X. He, F. Zeng, and Y. Deng, "Specifing Software Architectural Connectors in SAM," *Proc. 11th Int'l Conf. Software Eng. and Knowledge Eng.,* 1999.
[24] R.A. Kemmerer, "Analyzing Encryption Protocols Using Formal Verification Techniques," *IEEE J. Selected Areas in Comm.,* vol. 7, no. 4, pp. 448-457, 1989.
[25] C. Lai et al., "User Authentication and Authorization in the Java Platform," *Proc. Ann. Computer Security Applications Conf.,* pp. 285-290, 1999.
[26] G. Lowe, "An Attack on the Needham-Schroeder Public-Key Protocol," *Information Processing Letters,* 1995.
[27] C. Meadows, "Applying Formal Methods to the Analysis of a Key Management Protocol," *J. Computer Security,* pp. 5-36, 1992.
[28] B. Meyer, *Object—Oriented Construction.* Prentice Hall, 1988.
[29] T.J. Mowbray and W.A. Ruh, *Inside CORBA - Distributed Object Standards and Applications.* Addison-Wesley, 1997.
[30] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE,* vol. 77, no. 4, pp. 541-580, 1989.
[31] OMG. *CORBAservices: Common Object Services Specification, Security Service Specification.* Object Management Group, 1996.
[32] J.R. Putman, *Architecturing with RM-ODP.* Prentice Hall PTR, 2001.
[33] T. Riechmann and F.J. Hauck, "Meta Objects for Access Control: A Formal Model for Role-Based Principals," *Proc. New Security Paradigms Workshop,* pp. 30-38, 1998.
[34] J. Rumbaugh, M. Blaha, W. Premelani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design.* Prentice Hall, 1991.
[35] R. Sandhu et al., "Role-Based Access Control Models," *Computer,* vol. 29, no. 2, pp. 38-47, Feb. 1996.
[36] R. Sandhu and Q. Munawer, "How to Do Discretionary Access Control Using Roles," *Proc. ACM Workshop Role-Based Access Control,* pp. 47-54, 1998.
[37] S. Schneider, "Verifying Authentication Protocols in CSP," *IEEE Trans. Software Eng.,* vol. 24, no. 9, pp. 741-758, Sept. 1998.
[38] B. Thuraisingham and W. Ford, "Security Constraint Processing in a Multilevel Secure Distributed Database Management System," *IEEE Trans. Knowledge and Data Eng.,* vol. 7, no. 2, pp. 274-293, Apr. 1995.
[39] J. Wang and Y. Deng, "Incremental Modeling and Verification of Flexible Manufacturing Systems," *J. Intelligent Manufacturing,* vol. 10, no. 6, pp. 485-502, 1999.
[40] J. Wang, X. He, and Y. Deng, "Introducing Software Architecture Specification and Analysis in SAM through an Example," *Information and Software Technology,* vol. 41, no. 7, pp. 451-467, 1999.
[41] J. Wang, G. Xu, and Y. Deng, "Reduction Rules for Components in SAM," *Proc. Fifth Int'l Conf. Integrated Design and Process Technology,* 2000.

[42] J. Warmer and A. Kleppe, *The Object Constraint Language—Precise Modeling with UML.* Addison-Wesley, 1999.
[43] T.Y.C. Woo and S.S. Lam, "Designing a Distributed Authorization Service," *Proc. IEEE INFOCOM,* 1998.
[44] M.E. Zurko et al., "A User-Centered, Modular Authorization Service Built on an RBAC Foundation," *Proc. Ann. Computer Security Applications Conf.,* 1998.

Yi Deng received the PhD degree in computer science from the University of Pittsburgh in 1992. He is currently the director of the School of Computer Science at Florida Interanational University at Miami. From August 2000 to May 2002, he served as the managing director of the Embedded Software Center at the University of Texas at Dallas (UTD), a joint R&D consortium between UTD and industry dedicated to advanced software engineering technology for embedded software systems, and a tenured associate professor of computer science. Prior to joining UTD, he was at Florida International University (FIU) the State University of Florida at Miami, where he was the director of the Center for Advanced Distributed System Engineering (CADSE), a university research center designated by the Florida Board of Regents, and an associate professor of computer science. His research interests include component-based software engineering, software architecture, formal methods for complex systems, CORBA, and embedded systems. He has published extensively in these areas. He has been the PI/Co-PI of a number of research grants/contracts from various US federal agencies, such as US National Science Foundation, US Air Force Office of Scientific Research, US Air Force Research Laboratory, National Areonautics and Space Administration, as well as from industry. Dr. Deng is an editor for the *International Journal on Software Engineering and Knowledge Engineering*, a PC member and referee for many conferences and journals. He was the program committee cochair for 10th International Conference on Software Engineering and Knowledge Engineering (SEKE '98). He is a member of the IEEE and the ACM.

Jiacun Wang received the PhD in electrical and computer engineering from Nanjing University of Science and Technology (NUST), China, in 1989 and 1991, respectively. He is currently a member of the scientific staff with Nortel Networks in Richardson, Texas. Prior to joining Nortel, he was a senior research associate at the Center for Advanced Distributed Systems Engineering in the School of Computer Science, Florida International University (FIU). He was an associate professor at NUST. His research interests include software engineering, discrete event systems, formal methods, and distributed information systems. He has published more than 40 research papers in journals and conferences. He was a member of the program committee for the 1994 International Conference on Electronics and Information Technology, Beijing, China, a member of the program committee for the 1997 IEEE Conference on Systems, Man, and Cybernetics, Orlando, Florida, and a member of the program committee for the 1998 IEEE Conference on Systems, Man and Cybernetics, San Diego, California. Dr. Wang is a senior member of the IEEE.

Jeffrey J.P. Tsai received the PhD degree in computer science from Northwestern University, Evanston, Illinois. He is a professor in the Department of Electrical Engineering and Computer Science at the University of Illinois at Chicago, where he is also the director of the Distributed Real-Time Intelligent Systems Laboratory. He coauthored *Knowledge-Based Software Development for Real-Time Distributed Systems* (World Scientific, 1993), *Distributed Real-Time Systems: Monitoring, Visualization, Debugging, and Analysis* (John Wiley & Sons, Inc., 1996), *Compositional Verification of Concurrent and Real-Time Systems* (Kluwer, 2001), coedited *Monitoring and Debugging Distributed Real-Time Systems* (IEEE/CS Press, 1995), and has published extensively in the areas of knowledge-based software engineering, software architecture, requirements engineering, formal methods, agent-based systems, and distributed real-time systems. Dr. Tsai was the recipient of a University Scholar Award from the University of Illinois in 1994 and was presented a Technical Achievement Award from the IEEE Computer Society in 1997. He is currently the co-editor-in-chief of the *International Journal of Artificial Intelligence Tools*. He is also an editor of the *Annals of Software Engineering, the International Journal of Software Engineering and Knowledge Engineering*, and the *International Journal of Systems Integration*. He is a fellow of the IEEE, the AAAS, and the SDPS.

Konstantin Beznosov received the PhD in computer science from Florida International University (FIU) in 2000. He is currently a security architect at Concept Five Technologies, a premier e-business solutions provider. Prior to joining the company, he had been with Florida International University-the State University of Florida at Miami, where he was a senior research associate at the Center for Advanced Distributed System Engineering (CADSE), a university research center designated by the Florida Board of Regents, and a graduate student at the School of Computer Science. His research and professional interests include security of distributed enterprise application systems, component-based software engineering, software architecture, and middleware systems. He has a number of publications in these areas. Dr. Beznosov has been a PC member of the Distributed Objects and Components Security (DOCSec) Workshop. He is a member of the IEEE and the ACM