A Survey of Modelling and Analysis Approaches for Architecting Secure Software Systems

Lirong Dai[1] and Kendra Cooper[2]
(Corresponding author: Lirong Dai)

Department of Computer Science and Software Engineering, Seattle University[1]
P.O. 222000, Seattle, WA, USA, 98122 (Email: daia@seattleu.edu)
Department of Computer Science The University of Texas at Dallas[2]
MS 31, P.O. 830688, Richardson, TX, USA, 75083 (Email: kcooper@utdallas.edu)

(Received Dec. 5, 2005; revised and accepted Jan. 10 & Mar. 3, 2006)

Abstract

There has been a growing interest in investigating methodologies to support the development of secure systems in the software engineering research community. Recently, much attention has been focused on the modelling and analysis of security properties for systems at the software architecture design level. The potential benefits of this architecture level work are substantial: security flaws can be detected and removed earlier in the software development life-cycle. This reduces development time, reduces development cost, and improves the quality of the resulting system.

As a result of this attention, a wide variety of approaches have been proposed in the literature. At this point, a survey for researchers involved in the problem of systematically modelling and analyzing software architecture design that have security properties would be of value to the community. This paper presents such a survey; it includes a discussion of semi-formal, formal, integrated semi-formal and formal, and aspect-oriented approaches. Comparison criteria are defined including: the kinds of notations used to model the security properties (e.g., Petri nets, temporal logic, etc.), whether the approach supports the manual or automated analysis of security properties, the specific security property modelled (e.g., authentication, role-based access control, etc.), and the kind of example system that has been used to illustrate the approach (information, distributed, etc.).

Keywords: Aspect-oriented, formal methods, security, software architecture, unified modelling language

1 Introduction

Systematical engineering security into software applications is an important and difficult problem [1, 11, 27] The importance of the problem can be seen from the number of security incidents reported to the Computer Emergency Readiness Team Coordination Center (CERT/CC) and their associated costs.The CERT/CC data from 2003 reports 137,529 incidents; the cost of electronic crimes is reported at 666 million dollars [7]. Most of these incidents, which can involve from one to thousands of sites, result from software vulnerabilities. The CERT/CC data indicate the number of these incidents continues to rise. The difficulty of the problem stems from its breadth, as it covers many areas such as authentication, auditing, authorization, confidentiality, integrity, and non-repudiation (security standard ISO 7498-2) [20], where authentication verifies the claimed identity of a user or provider, auditing ensures that user activity is properly recorded and reviewed, authorization moderates information use and provision, confidentiality means information is provided only for proper use as appropriate to the sensitivity of the information, integrity makes certain that information is used in ways that allow only necessary changes, and non-repudiation ensures the identity of a user is irrefutably verified and recorded as protection against their later denying participation. Each of these can be further categorized. For example, authentication may include peer entity authentication and data origin authentication; confidentiality may include connection confidentiality, connectionless confidentiality, selective field confidentiality, and traffic flow confidentiality, etc., as presented in Figure 1.

Security needs to be considered during each phase of the software development life-cycle, including requirements specification and analysis, architecture design, detailed design, implementation, testing, and deployment. The software architecture of a system is the structure of the system which comprises software elements, these elements' externally visible properties, where externally visible properties refer to their provided services, performance characteristics, fault handling, shared resource usage, and so on, and the relationships among those elements [2]. Software architecture focuses on designing

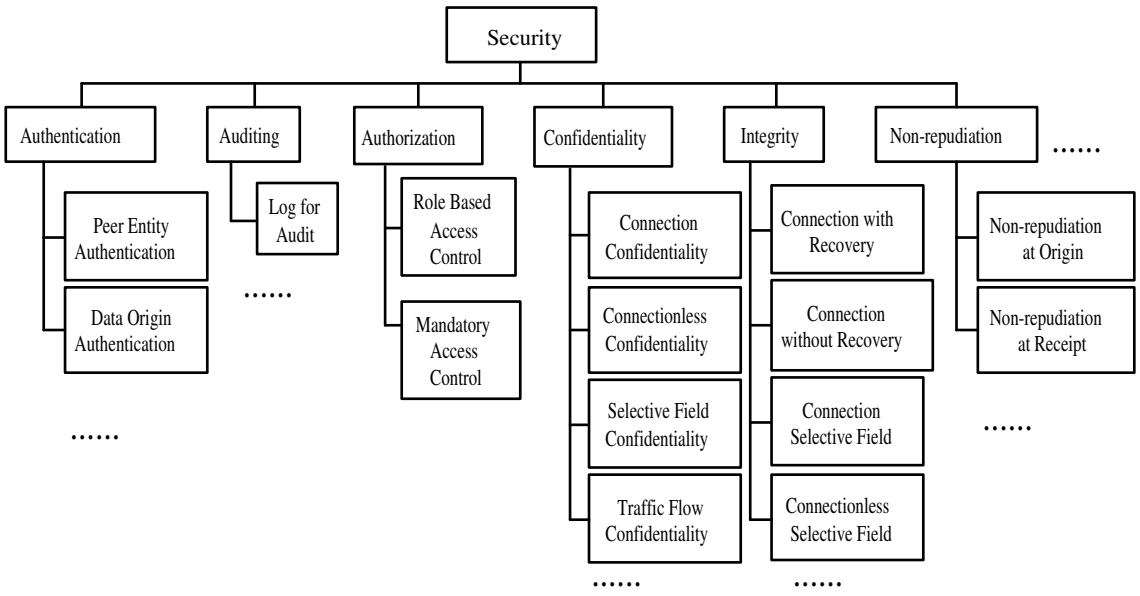

Figure 1: Elements of security

and specifying the overall system's gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives etc.

As the first design phase, it is widely recognized that decisions made at the architecture design stage have a strong impact on the quality of the final product [28]. Hence, to provide a positive impact, architecture designs, which reflect architectural decisions, should be analyzed so that design flaws can be detected and removed. Discovering and fixing defects at the architecture design stage is more cost- and time-effective compared to performing such work after the system is implemented, as fixing defects at the implementation stage would necessarily cause the revision and reconstruction of numerous design, implementation, and testing artifacts. Therefore, the architectural design and analysis of security properties is a very important step in the software development lifecycle. The architectural design of security properties enables the realization of a system's security non-functional requirements; the analysis of security properties provides architects with objective results to evaluate design alternatives.

Recently,numerous approaches have been proposed to support the modelling and analysis of security properties in software architecture designs. Given the diversity of the approaches, a survey paper is needed for researchers who are investigating the systematic design and analysis of security properties. Here, we present such a survey in which the approaches are classified into four broad categories: semi-formal (i.e., mainly using semi-formal methods in the approach), formal (i.e., mainly using formal methods in the approach), integrated semi-formal and formal (i.e., using a combination of semi-formal and formal methods), and aspect-oriented (i.e., security non-functional properties are modelled as aspects) approaches. The remainder of the paper is organized using these four categories. Section 2 presents a survey of semi-formal approaches, Section 3 presents formal approaches, Section 4 presents integrated semi-formal and formal approaches, and Section 5 presents aspect-oriented approaches. A discussion of the survey is in Section 6; conclusions are presented in Section 7.

2 Semi-formal Security Modelling and Analysis Approaches

Unified Modelling Language(UML)[26], a well-known notation, is a language for specifying, visualizing, constructing, and documenting designs of software systems. UML provides graphical notations to express the design of software systems with semi-formal syntax and semantics, and an associated language, the Object Constraint Language(OCL), for expressing logic constraints. UML contains two basic diagram types: structure diagrams and behavior diagrams. Structure diagrams depict the static structure of the elements in the system, including class, composite structure, component, deployment, object, and package diagrams. Behavior diagrams depict the dynamic behavior of the elements in the system, including activity, statechart, use case, communication, interaction overview, sequence, and timing diagrams. UML semantics are defined using a meta-model that is described in a semi-formal manner using three views: the abstract syntax, well-formedness rules, and modelling element seman-

tics. The abstract syntax is provided as a model described in a subset of UML, consisting of a UML class diagram and a supporting natural language description. The well-formedness rules are provided using the OCL and natural language (i.e., English). The UML meta-model is defined as one of the layers of a four-layer meta-modelling architecture, which includes meta-metamodel, meta-model, model, and user objects. This section presents several approaches, which use UML to model and analyze security non-functional properties.

2.1 MAC-UML Framework

The MAC-UML Framework [12] addresses the issue of incorporating Mandatory Access Control (MAC) into UML design artifacts, including use case, class, and sequence diagrams. The approach focuses on providing support for the definition of clearances and classifications for relevant UML elements.

In the approach, the concept of security assurance rules for a UML design is proposed. The basis of such security assurance rules is that UML use case diagrams, class diagrams, and sequence diagrams are abstracted into a set of UML elements. For example, there is a UML use case set $UC = \{uc_1, uc_2 \ldots\}$, UML actor set $AC = \{ac_1, ac_2 \ldots\}$, UML class set $C = \{c_1, c_2 \ldots\}$, and UML method set $M = \{m_1, m_2 \ldots\}$. Each UML element is assigned a clearance (CLR) or classification (CLS) from the partially ordered set $\sum = \{\perp = \sigma_1, \sigma_2 \ldots, \sigma_s\}$ where the order relation $\sigma_i < \sigma_j (i < j)$ means the security level σ_j has a higher security concern than that of σ_i. Notations $ac.CLR, uc.CLS, c.CLS_{min}, c.CLS_{max}$ and $m.CLS$ represent the CLR of actor ac, the CLS of use case uc, the min and max CLS of class c, and the CLS of method m, respectively. Then, three tiers of MAC security assurance rules are defined to assess the question of how to attain security in a design. Tier 1 security assurance rules represent the creation of use case diagrams with actors, use cases, actor-use case associations, actor and use case inheritance, and use case inclusion and extension relationships. For example, one of these security assurance rules can be interpreted as: For every actor ac_m that is associated with the use case uc_i (as a behavior of the application), the CLR of the actor ac_m must dominate the CLS of the use case uc_i. Formally, it can be represented as: $\forall$ actor ac_m and use case uc_i, ac_m is securely (MAC) associated with $uc_i \leftrightarrow ac_m CLR \geq uc_i.CLS$.

Correspondingly, security assurance rules for actor inheritance, use case inheritance, use case inclusion, and use case extension have also been defined. Tier 2 security assurance rules emphasis on defining the classes that are utilized by each use case: for a class c (intended) to be used in a sequence diagram to serve the goal (i.e., realize the functionality) of use case uc, the CLS of the uc must dominate the minimum CLS of c. Tier 3 security assurance rules are a refinement of Tier 2 security assurance rules to support method calls between the different entities (use case and objects) in a sequence diagram. Finally, algorithms are defined for assessing whether a UML design as a whole satisfies the security assurance rules by conducting a comprehensive analysis of the entire design.

The security problem addressed in the approach is mandatory access control (refer to Figure 1). The example system used is a Survey Institution which performs and manages public surveys. In the system, after the raw data of the survey is collected, staff with different privileges will manipulate the database, where senior person can add a survey header into the database, and another staff person (senior or junior staff) will add questions into that survey, and also have the ability to categorize questions and add a new question category if needed. However, there are some special questions that have more sensitive content, which only senior staff are allowed to perform data entry. The strength of the approach is that it bridges the gap between software engineering and an organization's security personnel. With the enforcement and assessment of three tiers of security assurance rules, the MAC capability can be incorporated into a UML design, where access violations through inheritance, inclusion, and extension can be detected. The approach can be applied to systems where MAC is one of the priority concerns. Such systems can be a distributed system, or an information system. The limitation of the approach is: since security assurance rules in the approach are only explored on a subset of UML diagrams (i.e., use case, class, and sequence diagram), the approach is only applicable to a system design that uses these three kinds of diagrams. In addition, security analysis of the approach is based on the semi-formal UML, where the relationship inheritance, inclusion, and extension are not formally defined. To obtain more rigorous analysis results, a formal specification of the system design is desired.

2.2 SecureUML

SecureUML [23] is a modelling language that defines a vocabulary for annotating UML based models with information relevant to access control. It is based on the role-based access control model (RBAC), with additional support for specifying authorization constraints. SecureUML defines a vocabulary for expressing different aspects of access control, like roles, role permissions, and user-role assignments. Due to its general access-control model and extensibility, SecureUML is well suited for business analysis as well as design models for different technologies. The structure of the modelling language conforms to the reference model for model driven systems. Model-driven software development is an approach where software systems are defined using models and constructed, at least in part, automatically from these models. A system can be modelled at different levels of abstraction or from different perspectives. The syntax of every model is defined by a meta-model.

The SecureUML meta-model is defined as an extension of the UML meta-model. The concepts of RBAC are represented directly as meta-model types, including

User, Role and Permission as well as relations between these types. Instead of defining a dedicated meta-model type to represent protected sources, every UML model element is allowed to take the role of a protected resource. Additionally, the type ResourceSet is introduced, which represents a user defined set of model elements. A Permission is a relation object connecting a role to a ModelElement or a ResourceSet. The semantics of a Permission are defined by the ActionType elements used to classify the Permission. Every ActionType represents a class of security relevant operations on a particular type of protected resource, for example, a method with the security relevant action executes, or an attribute with the action changes. ActionTypes give the developer a vocabulary to express permissions at a level close to the domain vocabulary. The set of ActionTypes available in the language can be freely defined using ResourceType elements. A ResourceType defines all action types available for a particular meta-model type. The connection to the meta-model type is represented by the attribute BaseClass, which holds the name of a type or a stereotype. The set of resource types and their action types, and the definition of their semantics on a particular platform, define the resource type model for the platform. An AuthorizationConstraint is a part of the access control policy of an application. It expresses a precondition imposed on every call to an operation of a particular resource, which usually depends on the dynamic state of the resource, the current call, or the environment. AuthorizationConstraint is derived from the UML core type Constraint. Such a constraint is attached either directly or indirectly, via a Permission, to a particular model element representing a protected resource.

The security problem addressed in the approach is role-based access control (refer to Figure 1). Enterprise JavaBeans (EJB) has been used in the approach. EJB is used in the industry for developing distributed systems. It is an industry standard with strong security support, which is implemented by a large number of application servers. The access control model of EJB is RBAC, where the protected resources are the methods accessible by the interfaces of an EJB. An access control policy is mainly realized by using declarative access control. This means that the access control policy is configured in the deployment descriptors of an EJB component. The security subsystem of the EJB environment is responsible for enforcing this policy on behalf of the components. The strength of the approach is that it developed a modelling language to build on the access control model of RBAC, with additional support for specifying authorization constraints in the context of a model-driven software development process to generate access control infrastructures. The approach helps to realize the RBAC capabilities in UML. The approach is suitable for distributed systems that incorporate RBAC model, such as an online banking system. However, the limitation of the approach is: currently, the approach only focuses on UML static design models, which are relatively close to the implementation. It is worth considering whether the efficiency of the development process of secure applications can be improved by annotating models at a higher level of abstraction (e.g. analysis) or by annotating dynamic models, e.g. UML state machines.

2.3 Separating Modelling of Application and Security Concerns

Separating Modelling of Application and Security Concerns (SMASC) is an approach proposed to model system's functional requirements separately from security requirements using UML use case, class, and object collaboration diagrams [15]. The motivation of the approach is to make a secure application system more maintainable with minimal impact on application concerns from changes to security concerns or vice versa. In the approach, the system is viewed through multiple views: a functional requirement view in UML use case diagram; a static view in UML class, and a dynamic view in UML object collaboration modelling. The system's functional requirements are modelled in "non-secure" business use cases. Security concerns are captured in security use cases and security objects, where security use cases are realized through message communications among security objects. The security use cases and objects can also be used in different application systems. Similarly, in the static model of the system, security concerns are separated from business concerns by modelling non-secure application classes separately from security service classes, and in the dynamic model, security concerns are separated from business concerns by modelling non-secure application objects separately from security objects. Therefore, the system's use cases, classes, and objects are divided into business layer and security layer. Security requirements are considered optional features meaning that if the security feature is required in a given application, then the appropriate security requirement condition is set to true, otherwise it is set to false. When the system requires security services, the security use cases are extended from the non-secure business use cases at extension points, which is a location where a use case extends another use case if the specified condition holds. The security use cases can have parameters, whose values are passed from the business use cases that they extend. Consequently, security classes and objects that realize the security feature can be added into the system's static and dynamic model.

The security problem addressed in the approach include integrity and non-repudiation (refer to Figure 1). The example system is an e-commerce application, where security concerns are separated from business concerns by modelling non-secure e-commerce application classes in the e-commerce business layer and secure classes in security layer. The strength of the approach is that it provides a way to capture security requirements in security use cases and encapsulate such requirements in security objects separately from the application requirements and objects. The approach reduces system complexity caused

by mixing security requirements with business application requirements, thus to increase the system's maintainability and components' reusability. However, one issue of the approach is that usually, security property is a pervasive property for a system which may crosscut many design model elements; therefore, clear separation of business and security model elements would not be a trivial task in this object-oriented approach.

3 Formal Security Modelling and Analysis Approaches

Formal methods [4] are referred to the variety of mathematical modelling techniques that are applicable to specify, develop, and verify computer system (software and hardware) design. A system's formal specification is the use of notations derived from formal logic to describe assumptions about the world in which a system operates, requirements that the system is to achieve, and a design to meet those requirements. Formal methods provide a way that a system can be formally specified whereby its desired properties can be reasoned about rigorously. Formal methods have been used to represent software architectures, where they provide a formal, abstract model for systems; thus, a means of describing and analyzing software architectures and architectural styles is available. This section presents several approaches that use formal methods to model and analyze security properties.

3.1 Software Architecture Model

Software Architecture Model (SAM) [17] is a general formal framework for visualizing, specifying, and analyzing software architectures. In SAM, a software architecture is visualized by a hierarchical set of boxes with ports connected by directed arcs. These boxes are called compositions. Each composition may contain other compositions. The bottom-level compositions are either components or connectors. Various constraints can be specified. This hierarchical model supports compositionality in both software architecture design and analysis, and thus facilitates scalability. Each component or connector is defined using Petri net. Thus, the internal logical structure of a component or connector is also visualized through the Petri net structure. Textually, a SAM software architecture is defined by a set of compositions $C = C_1, C_2, \ldots, C_k$ (each composition corresponds to a design level or the concept of sub-architecture). Each composition $C_i = \{C_{mi}, C_{ni}, C_{si}\}$ consists of a set C_{mi} of components, a set C_{ni} of connectors, and a set C_{si} of composition constraints. An element $C_{ij} = (S_{ij}, B_{ij})$, (either a component or a connector) in a composition C_i has a property specification S_{ij} (a temporal logic formula) and a behavior model B_{ij} (Petri net). Each composition constraint in C_{si} is also defined by a temporal logic formula. The interface of a behavior model B_{ij} consists of a set of places (called ports) that is the intersection among relevant components and connectors. Each property specification S_{ij} only uses the ports as its atomic propositions/predicates that are true in a given marking if they contain appropriate tokens. A composition constraint is defined as a property specification, however it often contains ports belonging to multiple components and/or connectors. A component C_{ij} can be refined into a lower-level composition C_l, which is defined by $h(C_{ij}) = C_l$(h is a hierarchical mapping relating compositions). The behavior of an overall software architecture is derived by composing the bottom-level behavior models of components and connectors. SAM provides both the modelling power and flexibility through the choice of different Petri net models. In SAM, software architecture properties are specified using a temporal logic. Depending on the given Petri net models, different temporal logics are used.

In [31], SAM is applied to support the formal design of software architecture for secure distributed systems. The security problem addressed is general information confidentiality (refer to Figure 1). The Petri net model used is the Predicate Transition Nets. The linear-time temporal logic is selected to formally specify security policies, the Chinese Wall policy, where *Basic objects* are individual items of information (e.g. files), each concerning a single corporation; *Company datasets* define groups of objects that refer to the same corporation; *Conflict of interest classes (COI)* define company datasets that refer to competing corporations. Subsequently, the definition of the sensitive dataset and secure distributed control architecture are provided. Finally, a new concept called the dependence relation for the Petri net model is defined, which gives source and sink of every work flow in the model and the dominating elements. Given an architecture model of a distributed system, a set of rules have been given to reconstruct the software architecture and enforce the security policy in the workflow level for a software architecture by examining the flow of sensitive datasets between tasks.

The strength of the approach is that it integrates two formalisms, Petri nets and temporal logic, to specify software architectures. The properties of the software architecture (e.g., information flow, deadlock, etc.) specified in Petri nets can be proved using temporal logic. Consequently, model checking techniques can be employed to realize the automated verification of software architecture properties. In addition, the approach provides a hierarchical architecture specification model, which enables iterative model checking in a bottom-up fashion. However, one issue of the approach is because of the limitation of model checking, the approach is generally not applicable to infinite state systems.

3.2 Multi Security Level Architecture

A modelling method for the Multilevel Security (MLS) architecture of the Defense Advanced Research Projects Agency's (DARPA's) Polymorphous Computing Architecture (PCA) program is proposed in [16]. PCA is a

multi-processor architecture that allows a processor to morph during operations to provide the best type of processor for the job at hand. The goal of MLS-PCA is to create a high assurance security infrastructure for multi-processor distributed applications, which means that the trusted aspects of the system needs to be verified, at a high level, under a certification program, such as the DoD's Trusted Computer Security Evaluation Criteria or its replacement, the Common Criteria (CC). In the proposed architecture, each single level Avionics Application Process (AAP) is front-ended by an Encryption Processing Element (EPE). All communication by an AAP must go through an EPE. All communication between EPEs is encrypted and authenticated. Keys are distributed to the EPEs by the Network Security Element (NSE) based on a security policy set up by mission control. The NSE enforces both Mandatory Access Control (MAC) and Discretionary Access Control (DAC). There is a unique key for each element of the security policy. For example, there is a key for each security level and compartment in the MAC security lattice, as well as for each pair in the DAC matrix. The NSE generates a session key between two AAPs by XORing the relevant policy keys with a one-time random key. The session key is then distributed to each of the AAP's EPE, where this session key must be distributed encrypted. All connections between two AAPs are simple. This allows a low level process to send information up to a high level process, but not vice versa. Another type of connection, called a coalition, that consists of AAPs at a common security level and using a common key is also allowed. In addition, the EPEs are also transparent to the AAPs, preventing the EPEs themselves from being used as a covert channel. High levels of evaluation require formal models and analysis. The selected formal method is Alloy [29]. The language is based on set theory and the first-order logic, similar to Z, with the standard set operators and quantifiers. A state is defined by sets and relationships among them. An operation will transform a state to a new state, i.e., the sets are modified. Alloy also allows the specification of invariants.

The security problem addressed in the approach is authentication (refer to Figure 1). With the use of formal method Alloy and its analysis tool in the approach, one can check the correctness of software architecture specification, using an inductive argument to claim that if an initial state is legal and all operations produce legal states, the system cannot be in an illegal state and the specification is correct. The approach also can be used to determine if a software architecture specification is over-constrained or under-constrained. The approach forces designers to look at the details of the architecture at an early stage of development, thus problems are detected and verification provides evidence that the requirements are maintained. One issue of the approach is: as in the analysis of an Alloy specification, a solution is obtained in a specific scope, therefore, the analysis of the approach is correct, but not complete.

3.3 Security Check

Security check is a technique proposed in [25] to entail taking small units of a system, putting them in a "security harness" that exercises relevant executions appropriately within the unit, and then model checking these tractable units. The technique is inspired by unit verification. The basic semantic framework used in the modelling is *discrete time labeled transition systems*. A discrete-time transition labeled transition system (DTLTS) is a tuple $< S, A \rightarrow, s_I >$ where: S is a set of states; A is a visible action set; $\rightarrow$ is the transition relation; and s_I is the start state. The distinguished elements τ and ι correspond to the *internal action and clock-tick* (or *idling*) action. A DTLTS encodes the operational behavior of a real time system. States may be seen as "configurations" the system may enter, while actions represent interactions with the system's environment that can cause state changes. The transition relation records which state changes may occur: if $< s, a, s' >$ is a transition relation, then the transition from state s to s' may take place whenever action a is enabled. τ is always enabled; other actions may require "permission" from the environment in order to be enabled. Also, transitions except those labeled by ι are assumed to be instantaneous. While unrealistic at a certain level, this assumption is mathematically convenient, and realistic systems, in which all transitions "take time", can be easily modelled. If a DTLTS satisfying the maximal progress property is in a state in which internal computation is possible, then no idling (clock ticks) can occur. DTLTSs model the passage of time and interactions with a system's environment. Finally, DTLTSs may be *minimized* by merging semantically equivalent but distinct states. The properties prove in *security check* are safety properties, including quasiliveness or bounded response which is a reasonable weakening of classical liveness. Both these classes of properties are inherent in any security property specification. While safety deals with properties of the form "nothing bad will happen" (i.e., the private key can never be revealed), liveness deals with assured-response or in a temporal setting bounded-response (i.e., the system will always respond in "t" time units even when under a DOS attack). *Security check* works by taking the property to be proved on the system and suitably crafting a "test process" based on that property (safety or liveness). The "unit", or modules inside the system to which the property is applicable, is isolated, all the behavior of the process not relevant to the property in question is "sealed" off and this transformed "unit" is first minimized and then run in parallel with the test process. Then it is checked if the test process terminates by emitting a pre-designated "good" or a "bad" transition. Depending on the transition the test process emitted it can be decided if a property is satisfied by the system or not.

The security problem addressed in the approach is to improve the intrusion detection capabilities for distributed system. The security check approach offers a

means of coping with state explosion of a system. The approach also enables to detect system vulnerabilities even when the attack behavior is not known. And for known attack patterns the approach can provide models of suspicious behavior which can then be used for intrusion detection at a later stage. One issue of the approach is since modelling checking techniques have been used; the approach is more suitable to finite state systems.

3.4 CVS-Server Security Architecture

An outline of a formal security analysis of a CVS-Server architecture is presented in [5]. The analysis is based on an abstract architecture (enforcing a role-based access control on the repository), which is refined to an implementation architecture based on the usual discretionary access control provided by the POSIX environment). The *Concurrent Versions System* (CVS) is a widely used tool for version management in many industrial software development projects, and plays a key role in open source projects usually performed by highly distributed teams. CVS provides a central database (the *repository*) and means to synchronize local modifications of partial copies (the *working copies*) with the global repository. CVS can be accessed via a network; this requires a security architecture establishing authentication, access control and non-repudiation. The proposed architecture aims to provide an improved CVS server, which overcomes the shortcomings of the default CVS server. The first aim of the work is to provide a particular configuration of a CVS server that enforces a role-based access control security policy. The second aim is to develop an "open CVS-Server architecture", where the repository is part of the usual shared file system of a local network and the server is a regular process on a machine in this network. While such an architecture has a number of advantages, the correctness and trustworthiness of the security mechanisms become a major concern, which leads to use formal methods to analyzing the access control problem of complex system technology and its configuration. The formal method Z has been chosen as the specification formalism. The modelling and theorem proving environment Isabelle/HOL-Z 2.0 is used, which is an integrated documentation, typechecking, and theorem proving environment for Z specifications.

The security problem addressed in the approach is role-based access control (refer to Figure 1), which is modelled and analyzed in the context of a CVS system. Therefore, the RBAC addressed in the approach is not generic to other applications.

4 Integrated Semi-formal and Formal Modelling and Analysis Approaches

This section presents the approaches that use a combination of semi-formal and formal methods to model and analyze security non-functional properties.

4.1 UML/Theorem Prover Approach

An extensible verification framework for verifying UML models for security requirements is presented in [21]. In particular, it includes various plug-ins performing different security analysis on models of the security extension UMLsec. Then an automated theorem prover binding is used to verify security properties of UMLsec models that make use of cryptography (such as cryptographic protocols). The UMLsec is an extension to UML that allows the expression of security relevant information within the diagrams in a system specification. UMLsec is defined in the form of a UML profile using the standard UML extension mechanism. The analysis routine in the verification framework supports the construction of automated requirements analysis tools for UML diagrams. The framework is connected to industrial CASE tools using data integration with XMI and allows convenient access to this data and to the human user. Advanced users of the UMLsec approach should be able to use this framework to implement verification routines for the constraints of self-defined stereotypes, in a way that allows them to concentrate on the verification logic (rather than on user interface issues). The usage of the framework proceeds as follows: the developer creates a model and stores it in the UML 1.4/XMI 1.2 file format; the file is imported by the UML verification framework into the internal Metadata Repository (MDR). MDR is an XMI-specific data-binding library that directly provides a representation of an XMI file on the abstraction level of a UML model through Java interfaces (JMI). This allows the developer to operate directly with UML concepts, such as classes, statecharts, and stereotypes. Each plug-in accesses the model through the JMI interfaces generated by the MDR library. The plug-ins may receive additional textual input and they may return both a UML model and textual output. The plug-ins include static and dynamic checkers. The static checker parses the model, verifies its static features, and delivers the results to the error analyzer. The dynamic checker translates the relevant fragments of the UML model into the automated theorem prover input language. The automated theorem prover is spawned by the UML framework as an external process; its results are delivered back to the error analyzer. The error analyzer uses the information received from the static checker and dynamic checker to produce a text report for the developer describing the problems found, and a modified UML model, where the errors found are visualized. Besides the automated theorem prover binding presented

in this paper there are other analysis plug-ins including a model-checker binding and plug-ins for simulation and test-sequence generation.

The security problem addressed in the approach is authentication (refer to Figure 1). The approach has been applied to a security-critical biometric system, where the control access to protected resources, such as a user's biometric reference data, needs to be ensured. Therefore, a cryptographic protocol is needed to protect the communication between the user biometric data reader and the host system. The protocol uses message counter in the transmission messages, thus to detect attacks. With the application of the approach, a flaw in the protocol, which allows attackers to misuse those message counters, has been detected. However, security properties that can be analyzed in the approach are limited to those which can be represented in first-order logic.

4.2 UML/Promela Approach

The UML/Promela approach [6] is proposed to investigate an appropriate template for security patterns that is tailored to meet the needs of secure systems development. In order to maximize comprehensibility, the well-known notation UML is used to represent structural and behavioral information. Furthermore, the verification of security properties is enabled by adding formal constraints to the patterns. The enhanced *security pattern template* presented herein contains additional information, including behavior, constraints, and related security principles, that addresses difficulties inherent to the development of security-critical systems. The security needs of a system depend highly on the environment in which the system is deployed. By introducing and connecting general security properties with a pattern's substance, the developer can gain security insight by reading and applying the pattern. Furthermore, behavioral information and security-related constraints are included in the security pattern template. The developer can use this information to check if a specific design and implementation of the pattern is consistent with the essential security properties. The augmented security pattern template includes fields' applicability, behavior, constraints, consequences, related security pattern, supported principles, and thus enhances the communication of security-specific knowledge that is related to a concrete application. Finally, a UML formalization framework that is developed to support the generation of formally specified models defined in terms of Promela [19], the language for the Spin model checker, thus to analyze the security pattern related requirements.

The security problem addressed in the approach is authorization (refer to Figure 1). The approach has been applied to an example system, where security properties, such as access violations from external requests to the system's internal entities, can be verified. These properties are instantiated in terms of linear time temporal logic to enable the analysis with Spin model checker. However, the approach currently focuses on the security property analysis against requirements. It needs to be extended in order to support the architecture level design and analysis of security properties.

5 Aspect-Oriented Security Modelling and Analysis Approaches

The principle of separation of concerns has long been a core principle in software engineering. It helps software engineers with managing the complexity of software system development. This principle refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concern (concept, goal, purpose, non-functional properties, etc.). However, many concerns of a system tend to crosscut many design elements at the design level; their implementation tends to crosscut many code units. Aspect-Oriented Software Development (AOSD) [13] technologies have been proposed to enable the modularization of such crosscutting concerns. In AOSD, a system's tangling concerns or pervasive properties are encapsulated in model element called an aspect. Subsequently, a weaving process is employed to compose core functionality model elements with these aspects, thereby generating an architecture design. This section presents aspect-oriented approaches which have been proposed to model and analyze security non-functional properties.

5.1 Aspect-Oriented Secure Application

An experience report based on developing security solutions for application software is presented in [30]. The programming language AspectJ [22] is used for this purpose. The engineering of application level security requirements are considered in this report, where the security concern covers many aspects, including authentication, auditing, authorization, confidentiality, integrity and non-repudiation. Security is a pervasive requirement for an application. Modularizing security concerns is a difficult task, and where and when to call a given security mechanism in an application has not been addressed adequately either. Furthermore, the crosscutting nature of security not only relates to the diversity of specific places where security mechanisms are to be called: some security mechanisms also require information that is not localized in the application. An example used in the report describes a Personal Information Management (PIM) system. A PIM system supports the human memory by keeping track of personal information, including a person's agenda, contact information of friends and business contacts, the tasks he has to fulfill, etc. A palm pilot is a typical example of a PIM system. In this system, the security requirement is the enforcement of access control. The design of this application is captured in a UML class diagram, where a class called PIMSystem is the center of the model. Through this class, the system can represent

and manage three different types of information: appointments, contacts and tasks. Two security access rules are: for appointments and tasks, the owner can invoke all their operations; other persons are only allowed to view them; for contacts, only owners can perform their operations. Finally, the report focuses on implementing these rules as aspects in AspectJ.

The security problem addressed in the approach is authorization (refer to Figure 1). The approach provides a way to implement authorization properties as crosscutting concerns in the aspect-oriented programming language AspectJ. However, it did not address the problem of aspect-oriented design of security properties.

5.2 Formal Design Analysis Framework

Formal Design Analysis Framework (FDAF) [8, 9, 10], is an aspect-oriented approach proposed to support the design and automated analysis of non-functional properties for software architectures. In the FDAF, non-functional properties are represented as aspects. At the architecture design level, aspect represents either a property that permeates all or part of a system, or a desired functionality that may affect the behavior of more than one architecture design elements, such as security aspects. Security aspects, including data origin authentication, role-based access control, and log for audit, have been defined in the FDAF. The definition of these security aspects uses UML diagrams. The definition for a security aspect includes its static view and dynamic view. The static view of a security aspect is defined using UML class diagram, presenting the attributes and operations need to be used in order to include the desired functionality in a system. It may also include OCL invariants, pre-conditions, and post-conditions regarding the weaving constraints of the security aspect. The dynamic view of a security aspect is defined in UML sequence diagram, describing the dynamic behavior of the security aspect, including the information about when and where to use the operations defined in the security aspect's static view. The FDAF proposes a UML extension to support the modelling of security aspects in UML. This extension assists architects in weaving an aspect into a design and updating an aspect in a design. The syntax and semantics for this UML extension have been defined. The FDAF uses a set of existing formal analysis tools to automatically analyze a UML architecture design that has been extended with security aspects. Architecture designs are documented using extended UML class diagram and swim lane activity diagram in the FDAF. The automated analysis of an extended UML architecture design in existing formal analysis tools is achieved by formalizing part of UML into a set of formal languages that have tool support. The translation into a formal language with existing tool support leverages a large body of work in the research community. The formalization approach used is the translational semantic approach [18]. In translational semantics, models specified in one language are given semantics by defining a mapping to a simpler language, or a language, which is better understood. Algorithms for mapping UML class and swim lane activity diagrams to a set of formal languages have been defined, verified with proofs, and implemented in the FDAF tool support, thus automated translation from UML to this set of formal languages are realized. Formal languages that UML can be formalized in the FDAF for the analysis of security properties include Promela [19] and Alloy [29], where Promela's analysis tool is used to analyze data origin authentication and log for audit security aspect design and Alloy's analysis tool is used to analyze role-based access control security aspect UML design.

The example system selected in the FDAF is the Domain Name System (DNS) [24], which is a real-time, distributed system. The security problem addressed includes data origin authentication, role-based access control, and log for audit (refer to Figure 1). There three security aspects have been modelled in the DNS, where data origin authentication is used to ensure the source of data, role-based access control is used to ensure the access control of the DNS database, and log for audit is used to log DNS messages. The strength of the approach is it integrates the well-known semi-formal notation UML and a set of existing formal notations into an aspect-oriented architectural framework to support the design and analysis of non-functional properties, such as security and performance. A limitation of the approach is that the analysis of a UML based architecture design uses existing formal tools, the limitations of these tools affect the analysis results provided in terms of accuracy, useful analysis data extraction, and interpretation of the results.

6 Discussion

A summary of the approaches in the survey is presented in Table 1. The approaches in this survey support modelling of one or more security properties at the architecture design level; many also support their automated analysis. The comparison criteria defined for this survey are: identifying the specific security property addressed (e.g., role-based access control, authentication, etc.), modelling notation(s) used, whether or not automated security property analysis is supported, and the kind of example system the approach has been applied to. Each of these criteria is useful, as the results can be used to guide the selection of an appropriate approach and identify possible areas for future research. For example, if one needs to model (but perhaps not analyze) role-based access control for a distributed system, then UML can be selected as the modelling notation. However, if there is a need for a more rigorous, automated analysis of the security property, then a formal method would be more suitable, such as Z or Alloy. If one needs to model confidentiality for information system, then Petri nets and temporal logic are candidate notations, as these have been already been successfully used to model this property. This is not to say

Table 1: Overview of approaches to design and analyze security properties

	Security Property	**Modelling Notations**	**Analysis**	**Example System**
UML-MAC Framework	Mandatory Access Control	UML	Supported	Information System
SecureUML	Role-Based Access Control	UML	Not supported	Distributed System Using EJB
SMASC	Integrity, Non-Repudiation	UML	Not supported	E-Commence Application System
Software Architecture Model	Confidentiality	Petri Nets, Temporal Logic	Not supported	Information System Using Chinese Wall Policy
Multi Level Security Architecture	Authentication	Alloy	Automated analysis	Real-Time System: MLS-PCA
Security Check	Intrusion detection	Discrete Time Labeled Transition System	Automated analysis	Distributed System
CVS-Server Security Architecture	Role-Based Access Control	Z	Automated analysis	Distributed system: CVS
UML/Theorem Prover Approach	Authentication	UML, First-Order Logic	Automated analysis	Biometric system
UML/Promela Approach	Authorization	UML, Liner Time Temporal Logic	Automated analysis	Distributed System
Aspect Oriented Secure Application	Authorization	UML	Not supported	Information System: Personal Information Management System
FDAF	Data Origin Authentication, Role-Based Access Control, Audit	UML, Formal languages (Promela, Alloy)	Automated analysis	Real-Time, Distributed System: Domain Name System

that other notations could not be used to model confidentiality. Actually, it opens a wide variety of possible future research topics that investigate the use of different notations, tailored notations, and perhaps identifying a set of notations that are suitable for modelling a comprehensive collection of security properties. It is also important to note that the validation of the approaches presented in the literature has typically been made using one example system. Additional validation of the approaches used to model and analyze security properties for the architecture design is necessary.

7 Conclusions

This paper presents a survey of current approaches in the literature that have been proposed to address the problem of modelling and analyzing security properties in architecture designs. These approaches are classified as semi-formal approaches, formal approaches, integrated semi-formal and formal approaches, and aspect-oriented approaches. The study showed that the well-known and easy-to-understand modelling language UML has been used in a variety approaches to model security non-functional properties. Furthermore, the automated analysis of non-functional properties needs formal methods, such as Architecture Description Languages, Petri nets, temporal logic, etc. Several proposed approaches have used a combination of semi-formal UML and formal methods in their work, thus to achieve the modelling efficiency provided by UML and the rigorous analysis provided by formal methods. Model checking and theorem prover based approaches have been also used as tools in the analysis.

Although a survey such as this is not exhaustive, we believe it will be useful to the research community involved in the architectural design and analysis of secure systems by providing a snap-shot of the current state-of-the-art. The wide variety of notations used indicates research is needed to investigate the use of different notations, tailored notations, and perhaps identifying a set of notations that are suitable for modelling and analyzing a comprehensive collection of security properties in software architectures.

References

[1] B. Arbaugh, "Security: Technical, social, and legal challenges," *Computer*, vol. 35, issue 2, pp. 109-111, Feb. 2002.

[2] S. Balsamo, M. Marzolla, and A. D. Marco, "Experimenting different software architectures performance techniques: A case study," in *Proceedings of the Fourth International Workshop on Software and Performance*, pp. 115-119, 2004.

[3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice (2nd Edition)*, Addison-Wesley, 2003.

[4] J. P. Bowen, R. W. Butler, D. L. Dill, R. L. Glass, D. Gries, and A. Hall, "An invitation to formal methods," *Computer*, vol. 29, issue 4, pp. 16-29, 1996.

[5] A. Brucker, and B. Wolff, "A case study of a formalized security architecture," *Electronics Notes in Theoretical Computer Science*, vol. 80, pp. 1-18, 2003.

[6] B. Cheng, S. Honrad, L. Campbell, and R. Wassermann, *Using Security Patterns to Model and Analyze Security Requirements*, Technical Report MSU-CSE-03-18, Department of Computer Science, Michigan State University, 2003.

[7] Computer Emergency Readiness Team Coordination Center (CERT/CC) 2004 E-Crime Watch, available at http://www.cert.org/about/ecrime.html

[8] K. Cooper, L. Dai, and Y. Deng, "Performance modelling and analysis of software architectures: an aspect-oriented UML based approach," *Journal of Science of Computer Programming, System and Software Architectures*, vol. 57, no. 1, pp. 89-108, July 2005.

[9] L. Dai, *Formal Design Analysis Framework: An Aspect-Oriented Architectural Framework*, The University of Texas at Dallas, Ph.D. Dissertation, 2005.

[10] L. Dai, K. Cooper, and E. Wong, "Modelling reusable security aspects for software architectures: a pattern driven approach," in *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering*, pp. 163-168, July 2005.

[11] N. Davis, W. Humphrey, S. T. Jr. Redwine, G. Zibulski, and G. McGraw, "Processes for producing secure software," *IEEE Security & Privacy Magazine*, vol. 2, no. 3, pp. 18-25, May-June 2004.

[12] T. Doan, S. Demurjian, T. C. Ting, and A. Ketterl, "MAC and UML for secure software design," in *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering*, pp. 75-85, 2004.

[13] R. Filman, T. Elrad, S. Clarke, and M. Aksit, *Aspect-Oriented Software Development*, Addison Wesley Professional, 2005.

[14] D. Garlan and M. Shaw, "An introduction to software architecture: advances in software engineering and knowledge engineering," *World Scientific Publishing*, vol. 1, pp 1-40, 1993.

[15] H. Gomaa and M. Eonsuk Shin, "Modelling complex systems by separating application and security concerns," in *Proceedings of the Ninth IEEE International Conference on Engineering Complex Computer Systems*, pp. 19-28, 2004.

[16] B. Hashii, "Lessons learned using alloy to formally specify MLS-PCA trusted security architecture," in *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering*, pp. 86-95, 2004.

[17] X. He, H. Yu, T. Shi, J. Ding, and Y. Deng, "Formally analyzing software architectural specifications using SAM," *Journal of Systems and Software*, vol. 71, no. 1-2, pp. 11-29, 2004.

[18] A. H. M. Hofstede, and H. A. Proper, "How to formalize It? Formalization principles for information system development methods," *Information and*

Software Technology, vol. 40, no. 10, pp. 519-540, 1998.

[19] G. J. Holzmann, *The Spin Model Checker: Primer and Reference Manual*, Addison-Wesley, 2003.

[20] ISO 17799 - *The Information Security Standard: Information technology*, Code of practice for information security management, 2000.

[21] J. Jurjens, "Sound methods and effective tools for model-based security engineering with UML," in *Proceedings of the 27th International Conference on Software Engineering*, pp. 322-331, 2005.

[22] R. Laddad, *AspectJ in Action: Practical Aspect-Oriented Programming*, Manning Publications, 2003.

[23] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A UML-based modelling language for model-driven security," in *Proceedings of the 5th International Conference on The Unified Modelling Language*, pp. 426-441, 2002.

[24] P. V. Mockapetris, *Domain Names - Concepts and Facilities*, IETF STD0013, November, 1987.

[25] A. Ray, "Security check: a formal yet practical framework for secure software architecture," in *Proceedings of the 2003 Workshop on New Security Paradigms*, pp. 59-65, 2003.

[26] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modelling Language Reference Manual*, 2nd Edition, Addison-Wesley, Reading, MA, 2004.

[27] J. Saltzer and M. Schroeder, "The protection of information in computer systems," *Proceedings of IEEE*, vol. 63, no. 9, pp. 1278-1308, 1975.

[28] M. Shaw and D. Garlan, *Software Architecture: Perspectives in an Emerging Discipline*, Prentice Hall, 1996.

[29] Software Design Group, the Alloy Analyzer, archived at http://alloy.mit.edu, 2002 - 2005.

[30] B. Win, W. Joosen, and f. Piessens, "Developing secure applications through aspect-oriented programming," *Aspect-Oriented Software Development (ISBN: 0321219767)*, pp. 633-651, 2005.

[31] H. Yu, X. He, S. Gao, and Y. Deng, "Formal software architecture design of secure distributed systems," in *Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'03)*, pp. 450-457, 2003.

Lirong Dai is an Assistant Professor in the department of Computer Science and Software Engineering at Seattle University. She received her Ph.D. in Computer Science in 2005 from The University of Texas at Dallas. Dr. Dai's research areas include software architecture development, realization, design, and analysis of non-functional requirements at the architecture design level, aspect-oriented software development (aspect modelling techniques, methods, tools, aspects in software architecture, etc.

Kendra Cooper is an Assistant Professor in Computer Science at the University of Texas at Dallas. She received her Ph.D. in Electrical and Computer Engineering in 2001 from The University of British Columbia. Dr. Cooper's research investigates requirements engineering and software architecture in component-based software engineering methodologies from agile-, aspect-, and goal-oriented perspectives using combinations of empirical studies and formal methods.