

Helping to Meet the Security Needs of Enterprises: Using FDAF to Build RBAC into Software Architectures

Lirong Dai and Kendra Cooper

Abstract—The vision, strategies, and goals of enterprises involve numerous security issues; these stem from legal and business concerns. For example, a financial organization, such as a bank, needs to ensure that employee and customer data are kept private and account balances for customers are not corrupted. Some of these needs may be realized in a collection of software applications such as employee payroll, employee performance review, and account reconciliation systems. The problem of effectively designing secure software systems to meet an organization's needs is a critical part of their success. This paper focuses on the problem of how to build security into a software architecture using the Formal Design Analysis Framework (FDAF). FDAF is an aspect-oriented approach that supports the design and analysis of non-functional properties for distributed, real-time systems. Particularly, an empirical study is presented to illustrate building Role-Based Access Control (RBAC), a design aspect in FDAF aspect repository, into the architecture for an online banking system. The RBAC aspect is adapted from the well-established RBAC security pattern. The study has also demonstrated how FDAF can help meet a system's enterprise level security requirements.

Index Terms—software architecture, aspect-oriented software development, security, enterprise requirements.

I. INTRODUCTION

The vision, goals, and strategies of enterprises involve numerous security issues, stemming from legal, business, and social concerns. For example, a financial enterprise, such as a bank, needs to ensure that employee and customer data are kept private, account balances for customers are not corrupted, accounts can only be accessed by authorized people, etc. The importance of the problem to enterprises 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 [5]. 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.

An overview of the enterprise level requirements (goals) and their realizations by software level requirements and software architecture is presented in Figure 1. The enterprise requirements reflect the organization's goals that will move (part of) the enterprise from where it is now to where the stakeholders want it to be. The goals are generally combinations or functional and non-functional goals (refer to

Figure 1). The challenges in accomplishing this include the involvement of numerous, diverse stakeholders with different needs from parts of the enterprise, or sub-enterprises, such as geographically distributed teams, departments, facilities, or subsidiaries. Part of the difficulty in achieving this for security requirements stems from the breadth of the problem, as security covers many areas (authentication, auditing, authorization, confidentiality, integrity, and non-repudiation as in security standard ISO 7498-2 [13]) and involves legal, business, and social issues [1]. Requirements engineering techniques are available in the literature for early requirements, including KAOS [7], Tropos [9], and the NFR Framework[4].

These early requirements are realized by various interacting sub-organizations in the enterprise using combinations of manual and automated workflow processes (refer to Figure 1) [12]. The need for more efficient, automated processes often drives the development or purchase of new or modified software systems. As it is not feasible to automate every workflow, specific functional and non-functional enterprise requirements are selected for realization in a software system, which could be built in-house, outsourced, or purchased. The enterprise requirements to be built are refined into software requirements; a software architecture is designed to realize the software requirements. Subsequent activities include defining the detailed design model and implementing the system.

In this work, we focus on the problem of how to build security into a software architecture to help meet enterprise level security requirements. In particular, we demonstrate building Role-Based Access Control into the architecture for an online banking system using the Formal Design Analysis Framework [6]. FDAF is an aspect-oriented architectural framework that supports the systematic modeling and automated analysis of non-functional properties, including security, using a combination of an extended UML and a set of existing formal methods. In FDAF, reusable security aspects are defined in UML; they are derived from security patterns. FDAF aspect definition provides concrete information regarding the use of security aspects, including explicit definition of attributes and aspect operations for realizing the security mechanisms (i.e., the interactions between the aspect and the application the aspect is woven into). A detailed survey of related approaches used to design and analyze security properties, including [3][14], is available

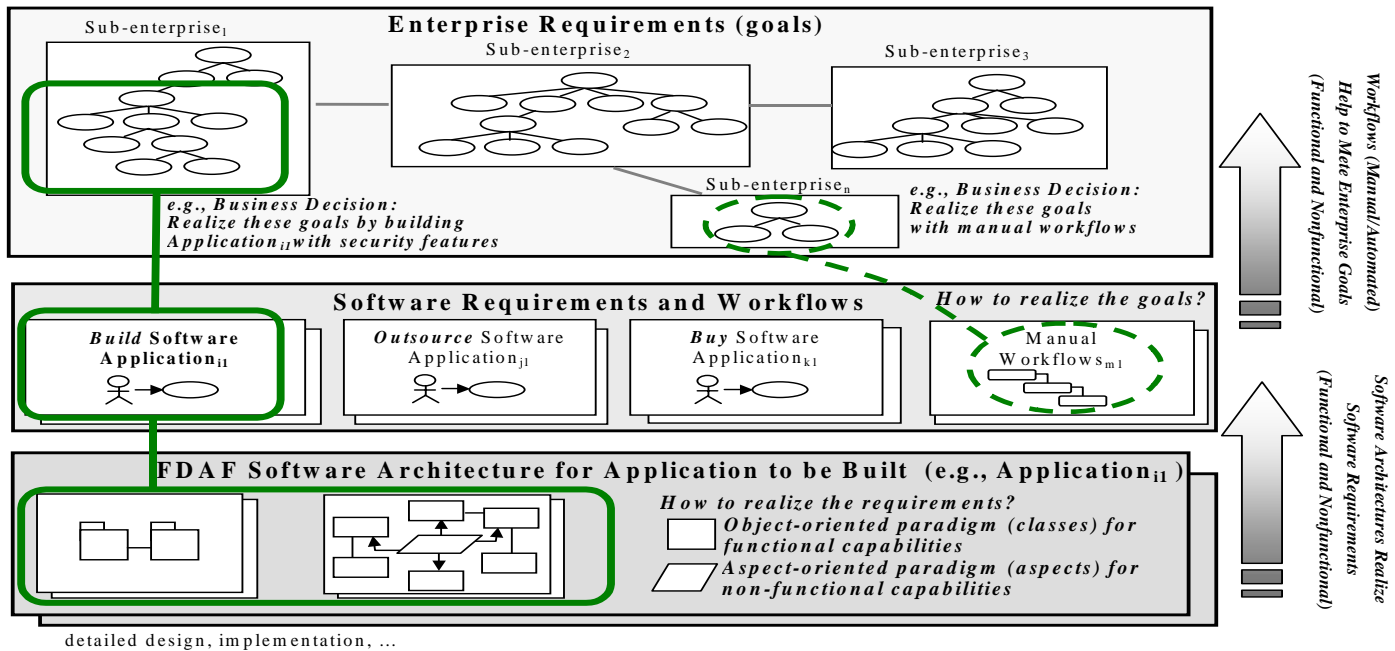


Figure 1 Overview of Enterprise Requirements and their Realizations

in [6]. Here, we briefly introduce the aspectual, or crosscutting, nature of security properties and discuss security patterns, as both are key features in FDAF. An overview of FDAF is presented in Section 2.

Security properties are pervasive properties of an application, as it is difficult to encapsulate many security properties into one part of the design. Such properties are referred as crosscutting properties. The new design discipline, Aspect-oriented software development (AOSD) [8], has been considered as a suitable solution to address the design of crosscutting concerns. AOSD is the adaptation of aspect-oriented programming principles to the design phase. Aspect-oriented programming techniques provide linguistic mechanisms for separate expression of concerns (i.e., stakeholders' interests), along with implementation technologies for weaving these separate concerns into working systems. In AOSD, a system's tangling concerns (e.g., non-functional properties) are encapsulated in model element called an *aspect*. Subsequently, a *weaving* process is employed to compose core functionality model elements (those realize the system's main functionalities) with these aspects, thereby generating a complete architecture design.

A security pattern [18] describes a particular recurring security problem that arises in specific contexts and presents well-proven generic schemes as solutions, thus key security problems are identified for novices and security problems are solved in a structured way [17]. Further, a security pattern enables security experts to identify, name, and discuss both problems and solutions more efficiently. However, the limitations of security patterns include: security patterns are mainly described in text (i.e., English), which are disconnected from design notations that system architects might actually use (i.e., UML); the use of security mechanism in a security pattern, such as *where* and *when* to call the given

security mechanism in an application, has not been addressed adequately; and finally, security pattern is relatively hard to be accepted by current security analysis tools.

The remainder of the paper is organized as follows. Section II presents an overview of FDAF. Section III presents the RBAC security pattern and security aspect. An illustration of modeling the RBAC security aspect for the example system, an online banking system, is presented in Section VI and is discussed in Section V. Conclusions and future work are presented in Section VI.

II. FORMAL DESIGN ANALYSIS FRAMEWORK

An overview of FDAF is presented in Figure 2. Users of FDAF include a variety of stakeholders, such as architects, designers, requirement engineers etc. The inputs of the framework are entities used by the stakeholders, including a system design documented in the UML and a requirement specification that includes the system's functional and non-functional requirements. The outputs of the framework are entities produced by the stakeholders, including a baselined UML design model, a set of formal aspect design models and the analysis results produced by formal analysis tools.

In FDAF, non-functional properties (e.g., performance, security, reliability, etc.) are represented as aspects in an architecture design. FDAF provides an aspect repository, where a collection of predefined aspects are available for architects to reuse. Architects can search the repository to select the appropriate aspect(s) according to the system's non-functional requirements. In the repository, aspects have been classified as two types: *property* and *substantive* aspects.

Property aspects are aspects that are used to describe properties of systems. Such aspects (e.g., response time aspect) normally use certain format of data value, e.g., numerical values, to express a system's property and they do

not trace to a specific implementation in the final product. Property aspects are primarily defined in UML stereotypes with tags, as these aspects are prescribed properties that permeate all or part of a system (e.g., meet a response time requirement). FDAF supports the definition for response time, resource utilization, and throughput performance aspects.

Substantive aspects are aspects that will be constructed in code. Substantive aspects are defined in UML as subsystems as these aspects are capabilities delivered by these subsystems (e.g., provide secure access control). FDAF supports the definition for data origin authentication, role-based access control, and log for audit security aspects.

To support modeling aspects, FDAF proposes an aspect-oriented UML extension. This is achieved by abstracting aspect-oriented implementation constructs, e.g., join points, advice, up to the design level. The syntax and semantics of the extension are defined in [6].

Once an aspect is captured in a UML based architecture design, FDAF uses a set of existing formal analysis tools to automatically analyze the extended design. 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 formal languages leverages a large body of work in the research community. The formalization approach used is the translational semantic approach [10]. In translational semantics, models specified in one language are given a semantics by defining a mapping to a simpler language, or a language which is better understood. Algorithms for mapping (part of) UML to a set of formal languages have been defined, verified with proofs, and implemented in FDAF tool support, thus automated translation from UML to the set of formal languages are

realized. The set of formal languages that UML can be formalized into include: Rapide ADL [15], Armani ADL [16], Æmilia ADL [2], Promela [11], and Alloy [19]. Each formal language has its own architectural modeling concentration (e.g., architectural connections, architectural constraints etc.), its syntax is considerably simpler and its semantics are well-understood. Rapide ADL's analysis tool is used to analyze response time aspect design; Armani ADL's analysis tool is used to analyze resource utilization aspect design; Æmilia ADL's analysis tool is used to analyze throughput aspect design; Promela's analysis tool is used to analyze data origin authentication and log for audit aspect design; and Alloy's analysis tool is used to analyze role-based access control aspect design, where the consistency of role-based access policies can be enforced. This has been documented in [6].

III. ROLE-BASED ACCESS CONTROL ASPECT

Access is the ability to do something with a computer resource (e.g., use, change, or view). Access control is the means by which the ability is explicitly enabled or restricted in some way (usually through physical and system-based controls). The RBAC model, as introduced in 1992 by Ferraiolo and Kuhn, has become the predominant model for advanced access control because it reduces the complexity and cost of security administration in large networked applications. The RBAC model is defined in terms of individual users being assigned to roles and permissions being assigned to roles. The benefits of RBAC include: low maintenance cost, increased efficiency, improved operational performance, reduced IT service, and administrative costs both internally and externally.

The RBAC security pattern [18] is defined as a reusable aspect in FDAF. The definition of a security aspect definition

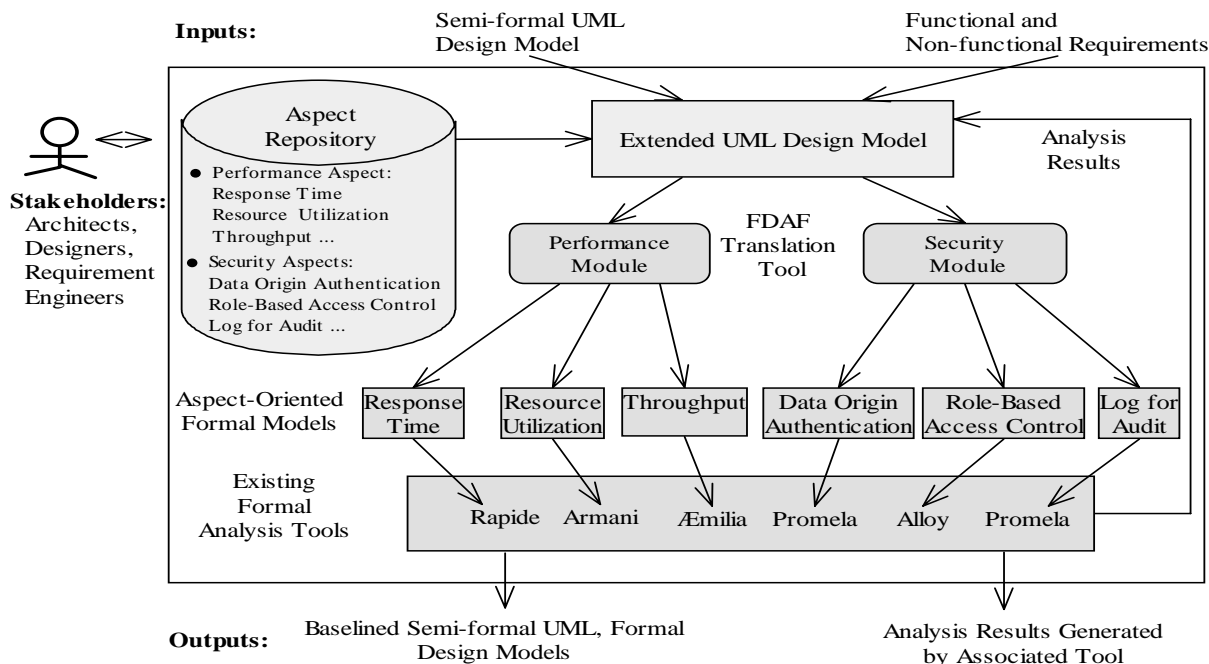


Figure 2. Formal Design Analysis Framework

may include a static view and a dynamic view: the static view of the aspect is captured in UML class diagram, where attributes and operations to realize the aspect capability are defined; the dynamic view of the aspect is captured in UML sequence diagram, where the execution behavior of the aspect is illustrated. Generally, the security pattern template [18] used to describe a security pattern includes these sections: name, abstract, problem, solution, issues, etc. Pattern adapting in FDAF focuses on adapting name, problem, and solution sections to aspect definitions: the name of a security pattern is adapted as the name of an aspect; entities (or participants) involved in the pattern are identified as attributes of the aspect; operations on those entities are identified as operations of the aspect; and finally, the pattern problem section can help identify the scenarios where the adapted aspect can be applied. With attributes and operations, the static view of an aspect is obtained. If there is any constraint on execution order among aspect operations or some of these operations need to be executed under certain condition, then a dynamic view of the aspect is necessary. This section presents the RBAC security pattern, followed by the definition of the RBAC aspect.

Role-Based Access Control Security Pattern. The RBAC security pattern is presented in Table 1. The pattern is adapted from [18].

Pattern Name	Role-Based Access Control
Abstract	Role-based access control specifies the control access to resources based only on the role of the subject.
Problem	Permissions for subjects accessing protected objects have to be described in a suitable way. A central authority should be responsible for granting the authorizations. A convenient administration of the authorizations should be guaranteed for a large number of subjects and objects.
Solution	Four concepts: user, role, right, and object. User and Role describe the registered users and the predefined roles, respectively. Users are assigned to roles, roles are given rights according to their functions. The association class Right defines the access types that a user within a role is authorized to apply to the protection object.

Table 1. Role-Based Access Control Security Pattern

Role-Based Access Control Security Aspect. Based on the solution provided by the RBAC security pattern, a static view of the RBAC in UML class diagram is presented in Figure 3. In FDAF, a parallelogram notation is used to present aspect information. The definition of the RBAC aspect includes five operations to manipulate the RBAC concepts as

defined in the RBAC security pattern: user, role, object, and right. Considering that in an access control environment, user, object and right may already exist in the system, the entity role has been identified as an attribute for the RBAC aspect. The operations of the RBAC aspect are:

- CreateRoleSession(): create a role session for users when they log in
- AssignRoles(): assign roles to users
- AssignRights(): assign rights to roles
- CheckRolesForActions(): verify whether a user with particular roles is authorized to take the actions or not
- DisableRoles(): disable roles

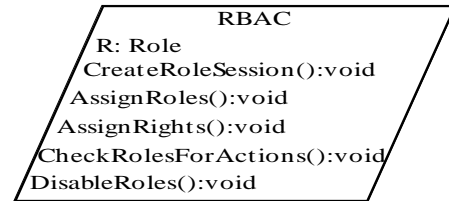


Figure 3. Role-Based Access Control Aspect

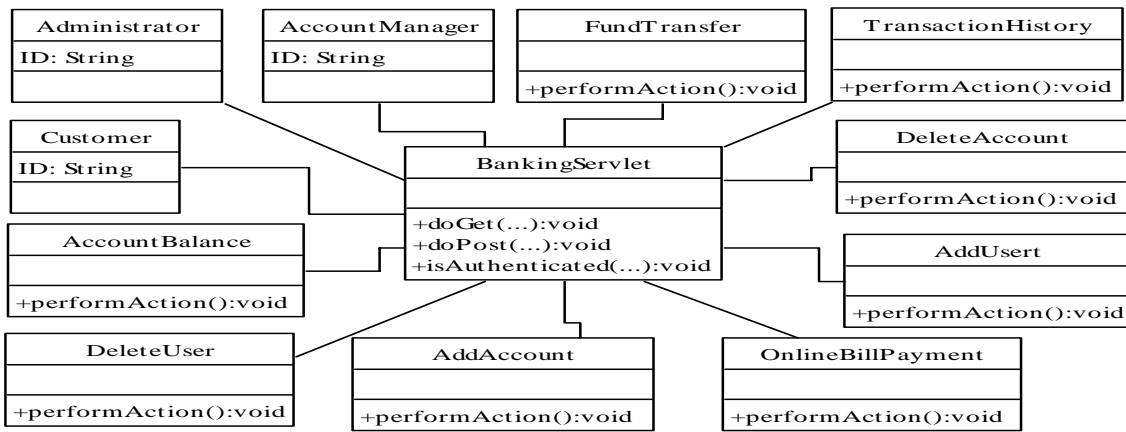
IV. ILLUSTRATION: ONLINE BANKING SYSTEM

This section presents a brief description of the enterprise requirements, system requirements (an online banking system), and illustrates how to weave the RBAC security aspect into the system's architecture design in a two-phase approach.

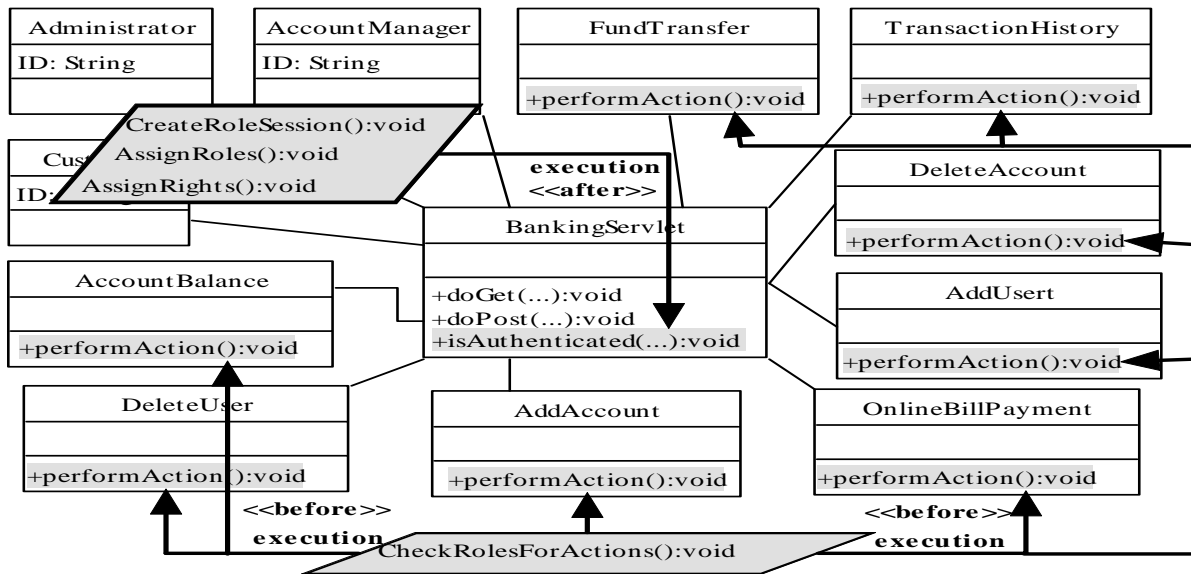
Summary of a Bank's Enterprise Requirements. A (partial) list of a Bank's enterprise requirements is provided here in textual form, due to space constraints:

- (1) The Bank shall increase its number of customers by 10% this fiscal year, by attracting customers who are seeking convenient, secure, accessible services.
- (2) The Bank shall increase customers accessibility to banking services by a) providing a new comprehensive On-line Banking System that is easy to use, secure, and scalable for the Bank's future growth b) extending Bank location hours on Saturdays, and c) extending the hours for the existing Telephone Banking System.
- (3) Extending the hours for the Bank locations and the existing Telephone Banking System are addressed with modifications to manual workflows. The new On-line Banking System can be developed in-house, outsourced, or purchased (e.g., if a turnkey system is available). We continue in the example using the decision that the system will be developed in-house.

Summary of the On-line Banking Software Requirements. A (partial) list of a Bank's software requirements is provided here in textual form, again due to space constraints:



a) Online Banking System Architecture Design without RBAC



b) Online Banking System Architecture Design with RBAC

Figure 4. The On-line Banking System Architecture: Before and After the Addition of the RBAC

- (1) The system's users include Customers, Account Managers (i.e., bank service representatives), and System Administrators.
- (2) Each kind of user can access different capabilities.
- (3) Each user needs to log in with his/her own user identifier and password to access the online banking service.
- (4) Customers can access their accounts stored in the bank's database through a PDA or a PC using an Internet browser.
- (5) Once a Customer is authorized, they can perform following actions on their accounts: account balance check, fund transfer, account history download, add, modify, or delete an on-line bill payee, pay a bill online, check interest rates, and submit a loan application.
- (6) Once an Account Managers is authorized, they can add, modify, and close accounts, and approve/deny loan applications.
- (7) Once a System Administrator is authorized, they can add, modify, and delete service representative or system administration accounts.

Phase One: Online Banking System without RBAC Security Aspect. In this phase, we designed the online banking system to meet its software requirements. The architecture design (a simplified version, tailored for the paper) is presented in Figure 4a. The architecture style selected is the Client-Server style. The architecture includes 12 components on the server side. The BankingServlet component provides services to interact with users. It is also the place where users get authorization checking information. Administrator, AccountManager, and Customer three components represent the three types of users of the system and they are maintained in the bank database; The rest of components represent actions (or rights) that can be taken by users, for example, view account balance (i.e., AccountBalance Component), transfer fund (i.e., FundTransfer Component), etc. The objects (e.g., accounts) of these actions are stored in the bank database and they are not presented in the architecture. Users may request for actions on objects; however, only if a user is eligible for the requested action, the action can be performed. For example, if a

customer requests for AddAccount action, this should be not allowed, as only AccountManager can perform the action. Without RBAC, this type of authorization checking is done on an individual user basis.

Phase Two: Online Banking System with RBAC Security Aspect. In this phase, we enforced the RBAC feature into the system (created in the first phase). The woven architecture design is presented in Figure 4b, where some operations in the base architecture design (i.e., from Figure 4a) are “connected” with RBAC aspect operations using a solid line. For example, the operation `isAuthenticated(...)` is connected with three aspect operations `CreateRoleSession()`, `AssignRoles()`, and `AssignRights()`; and the operation `performAction(...)` of each action component is “connected” with the aspect operation `CheckRolesForActions()`. Aspect operations are presented in a parallelogram notation.

In this phase, with the RBAC aspect enforced, every time after a user logs in, a role session for the user should be created, subsequently, roles for the user and rights for those roles are assigned. This means that the system should perform some additional actions (or capabilities) after the execution of the operation `isAuthenticated(...)`. These additional actions are encapsulated in RBAC aspect operations: `CreateRoleSession()`, `AssignRoles()`, and `AssignRights()`. Therefore, in the woven design (Figure 4b), a line with an arrow is used to “connect” the operation `isAuthenticated(...)` and those three aspect operations. This has indicated a crosscutting relationship between a UML base operation and aspect operations, and the texts “<<after>>” and “execution” describe two attributes of the crosscutting relationship, which means that *after* the *execution* of the operation `isAuthenticated(...)`, the system should perform additional actions: `CreateRoleSession()`, `AssignRoles()`, and `AssignRights()`. The rest of crosscutting relationship happens between the operation `performAction(...)` of each action component and the aspect operation `CheckRolesForActions()`. The texts “<<before>>” and “execution” indicate that with the RBAC aspect, *before* the *execution* of each `performAction(...)`, the system needs to verify the user’s roles using the aspect operation `CheckRolesForAction()`. The design presented in Figure 4b is the graphical presentation of FDAF UML extension to support aspect weaving. However, the meaning of the weaving is formally defined with aspect modeling elements proposed by this extension. These aspect modeling elements include Joinpoint, Pointcuts, Crosscutting relationship etc. They are abstractions of aspect-oriented programming concepts at the design level. A detailed description of FDAF aspect weaving and the aspect modeling elements’ syntax and semantics are available in [6].

V. DISCUSSION

This section presents the experience obtained from the empirical study of building RBAC into the architecture for the online banking system:

1) *Does FDAF security aspect help architects build RBAC security into a system, thus to bridge the gap between system architects and security professional? Is the RBAC security aspect definition adequate?*

Our empirical study showed that it is possible to make security capabilities as reusable aspects. However, here “reusable” does not mean that the aspect can be used without any assumptions and customization, but rather serving as a guidance of what a system should do in order to realize the desired security capabilities. For the instance of the RBAC aspect and the online banking system, in the architecture design of the first phase, we implemented each action as an individual class that was derived off of a base class, and the RBAC aspect would check upon execution of any subclass of this base class to see if the target object was allowed or restricted based upon the RBAC setup. If the same template were followed for a new application, the aspect we designed could be made generic. However, we also found out that if we were to have created an architecture design where the actions were grouped into sets, such as User actions, Administrator actions, and Account Manager actions, where each set of actions were in one class and each individual action were implemented as a method in its corresponding class, then the original design for core application would need to be changed in order to realize the RBAC feature. Hence, we have realized a refinement of the RBAC aspect definition is necessary. An attribute called “Assumption” for the aspect may facilitate the application of the aspect, where the attribute can be used to document related assumptions about the system. We have also found out that FDAF aspect provides guidance on what security operations are needed in a system, however, the definition could not provide the detailed implementation for each of these security operations, as the detailed implementation is tied to each application’s security requirements.

2) *Does building RBAC security into the software architecture help to meet an enterprise’s security requirements?*

In this study, we have demonstrated part of the refinement of a) enterprise level security requirements into software level RBAC security requirements and b) the subsequent refinement and inclusion of RBAC aspects into the software architecture. Based on this, we claim that part of the enterprise level security requirements has been met. If the example is broadened to include additional security enterprise and software requirements, then the architecture could also be extended and we could claim that more of the enterprise level security requirements have been met. However, this approach represents the technical issues involved meeting the enterprise level requirements with automated workflows. The manual workflows also need to be considered. Therefore, building security into the software architecture can help to meet an enterprise’s security requirements, but it does not mean that the enterprise requirement are guaranteed to be 100% met, since there are numerous other factors involved.

VI. CONCLUSION

Achieving enterprise level requirements (functional and non-functional) involves multiple levels of refinement. Enterprise goals can be realized with automated or manual workflows; automated workflows can be achieved by building, outsourcing, or buying software systems. For those going to be built, the software requirements (functional and non-functional) are realized by the software architecture.

The paper presents building Role-Based Access Control (RBAC) security into the software architecture for an on-line banking system using the Formal Design Analysis framework (FDAF). FDAF is a suitable design framework for this problem, as it is aspect-oriented and provides a repository of predefined reusable security aspects to use. The aspects are derived from established security patterns, which are developed by security experts. The architecture is developed in two phases: in the first phase, an online banking system without RBAC was designed using the client server style; in the second phase, the RBAC aspect was woven into the software architecture created in the first phase. The RBAC crosscut nine out of the twelve design elements introduced in the initial design.

We also used the study to evaluate the possibility of making security properties as reusable aspects. Our experience showed that it is feasible; however, we also found out that in order to reuse security aspects, certain assumptions about the system are necessary. For the example of the online banking system and RBAC aspect, there is an assumption about how the permissions of the system are organized in order to meet a system's security requirements. In turn, meeting the system's security requirements helps to meet the enterprise level security requirements.

There are several interesting directions for the future work. The first is to investigate extending FDAF to provide an aspect-oriented approach for modeling and analyzing enterprise and system level requirements, and their relationships to software architectures. This extension would provide a more comprehensive aspect-oriented modeling and analysis approach, encompassing the major activities in the early phases of software development. Second, the banking example can be extended with complete definitions of the enterprise level requirements, system requirements, and their traceability relationships, in order to support estimating the impact of changing, for example, a set of enterprise level requirements. Third, investigating how to model and analyze additional security aspects and their interactions. The NFR Framework is going to be used to systematically analyze the synergistic and conflicting relationships among the aspects. This is expected to be an interesting and challenging problem, as it necessitates a measurement, either quantitative or qualitative, of security capabilities (e.g., one encryption algorithm is in some way better or worse than another and by how much).

REFERENCES

- [1] Arbaugh, B., "Security: technical, social, and legal challenges", *Computer*, Volume 35, Issue 2, Feb. 2002, pp. 109 – 111.
- [2] Balsamo S., Bernardo M., and Simeoni M., "Combining stochastic process algebras and queuing networks for software architecture analysis", *Proceedings of ACM International Workshop on Software and Performance*, 2002, pp. 190-202.
- [3] Brucker A., and Wolff B., "A case study of a formalized security architecture", *Electronics Notes in Theoretical Computer Science* 80, 2003, pp. 1-18.
- [4] Chung, L., Nixon, B., Yu, E., and Mylopoulos, J., *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishing, 2000.
- [5] Computer Emergency Readiness Team Coordination Center (CERT/CC) 2004 E-Crime Watch, available at <http://www.cert.org/about/ecrime.html>.
- [6] Dai L., "Formal design analysis framework: an aspect-oriented architectural framework", The University of Texas at Dallas, Ph.D. Dissertation, 2005.
- [7] Darimont, R., Delor, E., Massonet, P. and van Lamsweerde, A., GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering, *ICSE 1997*: 612-613.
- [8] Filman R., Elrad T., Clarke S., and Aksit M., *Aspect-Oriented Software Development*. Addison Wesley Professional, 2005.
- [9] P. Giorgini, J. Mylopoulos, and R. Sebastiani, "Goal-Oriented Requirements Analysis and Reasoning in the Tropos Methodology", *Engineering Applications of Artificial Intelligence*, March 2005, 18 (2):159-171.
- [10] Hofstede A.H.M., and Proper H.A., "How to formalize It? Formalization principles for information system development methods", *Information and Software Technology*, 40(10), 1998, pp. 519 - 540.
- [11] Holzmann G.J., *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.
- [12] Hoogervorst, J., "Enterprise architecture: enabling integration, agility and change", In *International Journal of Cooperative Information Systems*, Sept. 2004, vol.13, no.3, pp. 213-33
- [13] ISO 17799 - The Information Security Standard: Information technology. Code of practice for information security management, 2000.
- [14] Lodderstedt T., Basin D., and Doser J., "SecureUML: A UML-based modeling language for model-driven security", *Proceedings of the 5th International Conference on The Unified Modeling Language*, 2002, pp. 426 - 441.
- [15] Luckham D.C., Kenney J.J., Augustin L.M., Vera J., Bryan D., and Mann W., "Specification and analysis of system architecture using Rapide", *IEEE Transactions on Software Engineering*, Vol. 21 Issue: 4, 1995, pp. 336 -354.
- [16] Monroe R.T., "Capturing software architecture design expertise with Armani", Technical Report No. CMU-CS-98-163, Carnegie Mellon University, School of Computer Science, 1998.
- [17] Riley J., "Bridging the gap between security and developers", *Computer Weekly*, April, 2004.
- [18] Security Pattern Repository, archived at www.securitypatterns.org, 2003.
- [19] Software Design Group, the Alloy Analyzer, archived at <http://alloy.mit.edu>, 2002 - 2005.