# DESIGNING HIGH INTEGRITY SYSTEMS USING ASPECTS

Geri Georg, Robert France, Indrakshi Ray
*Agilent Technologies, Colorado State University, Colorado State University*

Abstract: In this paper we show how design-level aspects can be used to develop high integrity systems. In our approach, a system designer must first identify the specific mechanisms required for high integrity systems. To support this activity we have developed an initial tabulation of different kinds of threats and the mechanisms used to prevent, detect, and recover from the related attacks and problems. Each mechanism can be modeled independently as an aspect. After the mechanisms are identified, the corresponding aspects are then woven in the appropriate order into the models of the essential system functionality to produce a model of a high integrity system.

Key words: Aspect, Integrity, Software design

## 1. INTRODUCTION

High integrity systems that carry out safety-critical and security-critical tasks require that close attention be paid to integrity concerns during software development. There is a growing awareness that the manner in which software is designed can have a significant impact on the security of the system [8]. Software developers need to consider integrity concerns when making architectural, logical, and physical (including technology-related) design decisions. For complex systems, developers also have to be concerned with other concerns, for example, distribution, and usability. A design technique that allows developers to model these concerns in a relatively independent manner, and then integrate (weave) them with a model of system functionality to produce a more comprehensive model of

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35693-8_16

M. Gertz (ed.), *Integrity and Internal Control in Information Systems V*

the system, can help pave the way for the development of systematic approaches to complex system development.

In this paper we propose and illustrate an aspect-oriented design (AOD) technique for designing a high integrity system. An aspect-oriented design consists of a primary model and one or more aspect models. An aspect encapsulates information related to a design concern (e.g., user authentication) that is spread across the design units of a primary design model (i.e., the concerns cross-cut the units of the primary design). Incorporating the aspects into a primary model is called weaving. An integrated model of the system can be obtained by weaving a primary model with the aspect models.

Treating high integrity concerns (e.g., security and fault-tolerance concerns) as aspects during design modeling has the following advantages: (1) Aspects allow one to understand and communicate high integrity concerns in their essential forms, rather than in terms of a specific application's behavior. (2) An aspect focuses on one concern, hence it is easier to model and understand its behavior. (3) The integrity aspects are potentially reusable across different systems. (4) Changes to the respective concerns are made in one place (aspects), and effected by weaving the aspects into the models of essential functionality. (5) The impact of high integrity concerns on system functionality can be analyzed by weaving the aspects representing the concerns into models of system functionality and analyzing the results. Analysis can reveal undesirable emergent behaviors and interference across the woven aspects at the design level. System designers are then better able to identify problems with the design of integrity mechanisms before they are implemented.

In order to realize the above benefits we define aspects as patterns expressed in terms of structures of *roles* called Role Models [11,12]. Roles define properties of the concerns represented by aspects. Model elements that have the properties specified in a role realize (or, can play) the role. In our approach, the model elements are UML constructs. Our work is based on UML because it is emerging as a *de facto* modeling standard.

The manner in which multiple aspects are woven into a primary model is determined by weaving strategies. A weaving strategy identifies integrity aspects based on the kinds of attacks and problems that are possible in a system and the mechanisms that can be used to detect, prevent, and recover from such attacks and problems. Each of these mechanisms can be modeled as an aspect. These aspects can then be woven with a model of system functionality to produce a design of a high integrity system. In this paper we illustrate how two aspects can be woven with a primary model.

The rest of the paper is organized as follows. Section 2 gives some background information pertaining to this work. Section 3 shows how integrity aspects can be specified using Role Models. Section 4 illustrates how two different security aspects (authentication and auditing) can be woven with a primary model. Section 5 concludes the paper.

# 2. BACKGROUND

## 2.1 Role Models

We use Role Models (see [11, 12]) to define aspects as patterns of model structures. Roles are used to define properties of aspects. A Role Model is a structure of roles, where a *role* defines properties that must be satisfied by conforming UML model elements. A UML model element (e.g., a class or an association) that has the properties specified in a role can *play the role*, that is, it *conforms to* (or realizes) the role. A UML model is said to conform to (or realize) a Role Model (i.e., is a realization) if (1) it contains model elements that conform to the roles in the Role Model and (2) the structure of the UML model is consistent with the structure characterized by the Role Model. Weaving an aspect defined by Role Models into a primary model is essentially a model transformation process in which a non-conforming primary model is transformed to a conforming model (i.e., a model that incorporates the aspect).

A design aspect (e.g., an integrity concern) can be modeled from a variety of perspectives. In this paper we focus on two aspect views: static and interaction views. An aspect's static view defines the structural properties of the aspect. The interaction view specifies the interaction patterns associated with the aspect. To model aspects from these views, we use specialized forms of two types of Role Models: *Static Role Models* (SRMs) and *Interaction Role Models* (IRMs) [12]. SRMs define patterns of UML static structural models (e.g. Class Diagrams patterns), while IRMs define UML interaction diagram patterns (e.g., Collaboration Diagrams patterns). An aspect definition typically consists of a single SRM and one or more IRMs.

### 2.1.1 An Overview of SRMs

An SRM consists of classifier and relationship roles. Each role has a *base* that restricts the type of UML construct that can play the role. An example of an SRM for authentication (an integrity concern) is shown in Figure 1

(details of the properties expressed in the roles are suppressed in this SRM). This SRM consists of four class roles: *Initiator*, *Target*, *IdentityRepository*, and *IdentityEntity*. The base of these roles is Class (as indicated by the *Class Role* stereotype), thus only UML class constructs can play these roles. The SRM also consists of three relationship roles (each with base *Association*) that characterize associations between classes playing the connected roles. Association roles have multiplicity constraints expressed in template forms (e.g. *[[n]]*) or as explicit sets of ranges (e.g., a UML multiplicity such as *2..n*). If a multiplicity (an explicit set of ranges) is shown on an association role end, then the multiplicity must appear on the ends of the realizing associations. Multiplicities in a realization of an association role containing these template multiplicities can be obtained by substituting values of *n* that satisfy the constraints associated with *n* (expressed using Object Constraint Language (OCL)).

Each role defines properties that conforming constructs must possess. Two types of properties can be specified in a SRM role: *Metamodel-level constraints* are well-formedness rules that constrain the form of UML constructs that can realize the role, and *Feature roles* characterize properties that must be expressed in the conforming model elements. Metamodel-level constraints are constraints over the UML metamodel expressed in OCL [24, 31]). For example, a metamodel-level constraint in a class role can constrain conforming classes to be concrete with a class multiplicity that cannot exceed a value *m*. In order to keep the diagrams simple in this paper, we do not show metamodel-level constraints. For examples of their use see [11].

Feature roles are associated only with classifier roles. There are two types of feature roles: *Structural roles* specify state-related properties that are realized by attributes or value-returning operations in a SRM role realization, and *behavioral roles* specify behaviors that can be realized by a single operation or method, or by a composition of operations. A feature role consists of a name, and an optional property specification expressed as *a constraint template* (omitted in the examples given in this paper). For example, each behavioral role (see [13]) is associated with pre- and post-constraint templates that, when instantiated, produce pre- and post-conditions of realizing operations.

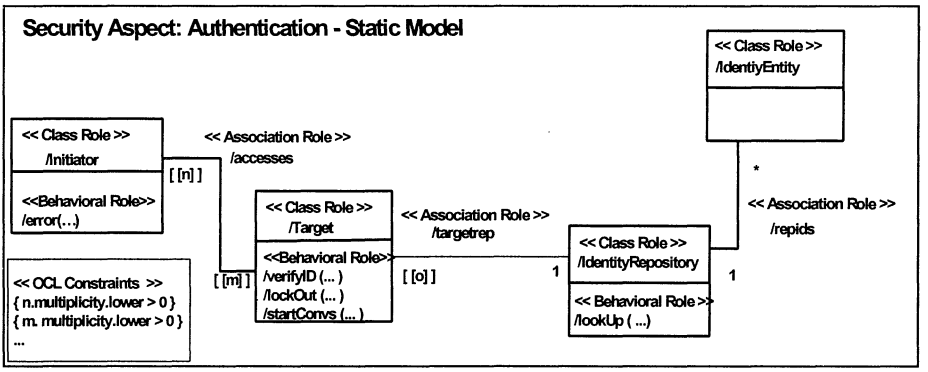


*Figure 1.* Static view of an authentication aspect.

### 2.1.2 An Overview of IRMs

UML interaction models (e.g., Collaboration and Sequence Diagrams) are used to specify the interactions between system parts. Interactions characterized by an aspect are defined using a template form of interaction models called Interaction Role Models (IRMs) [12]. An IRM consists of template forms of UML collaboration roles and messages. Instantiating these templates results in an Interaction Diagram. In this paper we use only the Collaboration Diagram template form of IRMs to describe the interaction pattern defined in an aspect. An example of an IRM for an authentication aspect is shown in Figure 2. This IRM is described in Section 3.

### 2.1.3 Generating Models Elements from Role Models

During aspect weaving new model elements may need to be created in order to incorporate the aspect into the original model. Conforming constructs are generated from an aspect's SRM as follows:

- Create an instance of the role base that satisfies the metamodel-level constraints.
- For each structural role, generate an attribute and associated constraints by substituting a name for the role name, and substituting conforming values for the template parameters of the constraint templates.
- For each behavioral role, generate an operation and associated constraints by substituting a name for the role name, and by substituting conforming values for the template parameters for the pre- and post-condition constraint templates.
- When multiplicity templates are given on association roles, association multiplicities can be obtained by substituting values that satisfy the constraints on the template parameters.

Interaction models can be produced from IRMs by substituting values for the template parameters that satisfy any constraints associated with the parameters.

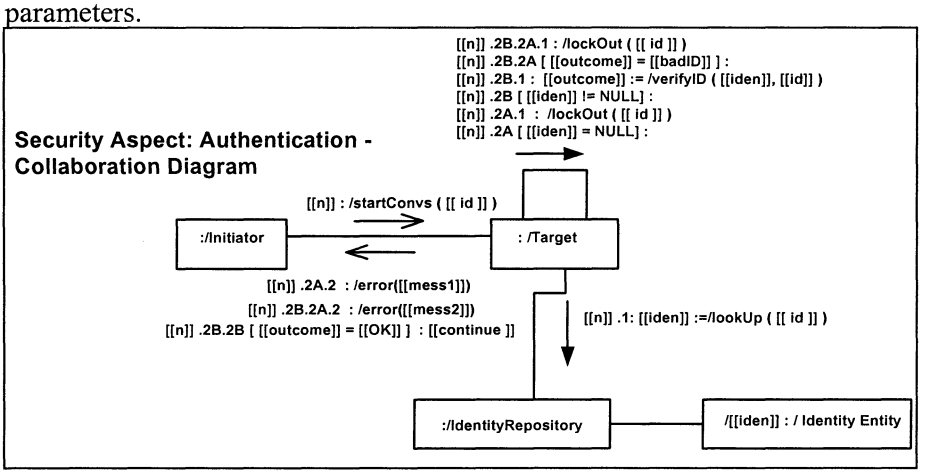


*Figure 2.* IRM for the authentication aspect

## 2.2 Definitions

In this section we present some definitions that we use in the rest of this paper. We begin with the definition of high integrity systems. A *high integrity system* is one that performs some safety-critical or security-critical operations. High integrity systems must be designed keeping in mind fault-tolerance and security issues. The three main objectives of high integrity systems are: (1) *Confidentiality* (sometimes termed privacy or secrecy): (i) protecting against unauthorized disclosure of information, (ii) protecting against accidental disclosure of information caused by system failures or user errors. (2) *Integrity*: (i) protecting against unauthorized modification of information, (ii) protecting against unintentional modification of information caused by system failures or user errors. (3) *Availability*: (i) protecting against unauthorized withholding of information or resources, (ii) protecting against failures of resources. Note that, secure systems also have these three objectives but the scope of these objectives focus only on unauthorized access. (For the definition of security objectives we refer the interested reader to [27].) High integrity systems, on the other hand, focus on both fault-tolerance and security issues for each of these objectives.

In the security domain *user* usually refers to the human user interacting with the computer system, *subject* is a computation entity, or executing thread, that performs actions on other entities, and *object* is an entity acted upon by a subject (not to be confused with the OO notion of an object). Typically not all subjects or users in a system can be trusted to maintain the

security objectives. A *trusted subject/user* is a subject/user that will not cause any security violations. A subject/user that may cause security violations is said to be *un-trusted.* Some objects contain critical information and others do not. The ones that contain critical information often need additional protection. A *sensitive object* is an object that contains sensitive information that should be protected from unauthorized access.

Finally, we define what we mean by attacks, threats, vulnerabilities, and integrity mechanisms. An *attack* is any action that compromises the security of information/resources owned by an organization. A *threat* is circumstances that have the potential to cause harm to a system. There are four threat categories: natural, environmental, human intentional and human unintentional. Examples of natural threats are natural disasters, such as, earthquake, volcano. Examples of environmental threats are power disruption, and smoke. Human intentional threats include unauthorized access, denial of service etc. Human unintentional threats include data loss, communication disconnections. Note that, while designing high integrity systems we assume that adequate protection is already in place for natural and environmental threats; we focus only on human intentional and human unintentional threats. High integrity systems must have secure mechanisms to protect against human intentional threats and fault-tolerant mechanisms to protect against human unintentional threats. *Vulnerability* is a weakness in the system that might be exploited to cause harm. A *problem* is an error or failure in the system. An *integrity mechanism* is a set of techniques that describe how the information and the resources can be protected. Mechanisms can be grouped into those that detect, prevent, or recover from attacks or problems. Authentication, access control, and encryption are examples of preventive mechanisms, auditing and intrusion tracking are detection mechanisms, and intrusion response is a recovery mechanism. *Authentication* is the process of verifying that the identity a user claims is indeed his true identity. *Auditing* is the process of recording the activities taking place in the system in an event log that is to be used for future analysis.

## 2.3 Integrity Mechanisms

The first step in designing a high integrity system is to identify the kinds of threats and the mechanisms that can be used to protect the system against such threats. Table 1 gives examples of attacks and problems, and the mechanisms that are commonly used to protect against them. (Note that this list is by no means exhaustive.) The table gives the type of attack/problem for each type of entity and the mechanisms used to protect against such attacks/problems. For example, consider row 3. This row states

that for un-trusted communication links the problem of lost communications can be solved by some handshaking protocol. Often times, the technology will dictate the presence or absence of problems. For example, when using TCP connections the problem of lost messages is not an issue.

Note that, in a real world scenario, these entities will not be acting alone but will act as a group. A group of entities will require a combination of the respective mechanisms. For example, a sensitive object may require encryption, authentication and access control to protect against unauthorized modification. When this sensitive object passes over un-trusted communication link we must incorporate the additional mechanisms needed for lost, spurious and corrupted communications.

In our AOD approach, a *weaving strategy* determines the aspects and constrains the manner in which they will be woven. Weaving strategies need to be developed from the kinds of threats or problems that can be expected in the system. For example, if the data passing over communication links in the user management system is not sensitive, eavesdropping may not be of concern. In this case, even if a communication link is un-trusted, encryptions are not needed and can be omitted from the weaving strategies.

*Table 1.* Attacks, Problems, and Solutions

| Entity | Type of Attack/Problem | Prevention, Detection, Recovery Mechanisms |
|---|---|---|
| Trusted communication link | Link error | Checksums |
| Trusted subject | User error | Auditing |
| Un-trusted communication link | Lost communications | Handshaking protocol, message retransmission |
| | Spurious or replayed communications | Authentication, use nonce, auditing, non-repudiation |
| | Corrupted communications | Checksums, encryption |
| | Eavesdropping | Encryption |
| Un-trusted subject | Un-authorized access | Authentication, access-control, auditing |
| Sensitive object | Un-authorized disclosure | Encryption, authentication, access-control, auditing |
| | Un-authorized modification | Authentication, access-control, encryption, checksums, auditing |
| | Un-authorized execution | Authentication access-control, auditing |

## 2.4 Related Work

There has been much work on aspect-oriented programming [4, 5, 20, 21, 25, 29] and a number of authors have tackled the problem of defining and weaving aspects above the programming language level (e.g., see [3, 7, 10, 13, 16, 23, 26, 28]). In the latter cases, aspect specifications can be viewed as template models, and they are generally woven by using regular expression to match existing model elements and aspect elements. Many aspect compositions essentially result in wrapping additional functionality around an existing model. The proper model factoring must already exist to apply the aspect, so it is conceivable that effort must be applied to re-factor existing models to correctly compose them with aspect models. The technique proposed in this paper supports a more flexible and rigorous approach to aspect definition and weaving in aspect-oriented design (AOD).

There has been some work on using the UML to model security concerns (e.g., see [1, 2, 6, 9, 12, 15, 17, 18, 19, 22, 30, 32]). Chan and Kwok [6] model a design pattern for security that addresses asset and functional distribution, vulnerability, threat, and impact of loss. UML stereotypes identify classes that have particular security needs due to their vulnerability either as assets or as a result of functional distribution. Jurjens [18, 19] models security mechanisms based on the multi-level classification of data in a system using an extended form of the UML called UMLsec. The UML tag extension mechanism is used to denote sensitive data. Statechart diagrams model the dynamic behavior of objects, and sequence diagrams are used to model protocols. Deployment diagrams are also used to model links between components across servers. UMLsec is fully described in a UML profile.

In a previous work [13] we show how aspects can be expressed as Role Models that define patterns of design semantics, and how aspects can be woven into designs expressed in terms of the UML. Our approach supports a more rigorous approach to the design and weaving of aspects, and results in better aspect reuse potential. A second work [14] is on how security aspects can be modeled independently and then be woven into models of essential functionality. By providing good support for separation of concerns during the design phase, the complexity of designing large systems can be better managed. Our work can also take advantage of UML security extensions: the weaving process can be designed to produce extended forms of the UML that reflect the properties expressed in the aspects in a more direct manner.

This work extends our previous work [14] in the following manner. In this work we model two independent integrity mechanisms as aspects and show how these two aspects (authentication and auditing) can be woven in with a model of system functionality (the primary model). We also show that

the order in which the aspects are woven also plays a critical role. Incorrect weaving order will produce an incomplete woven model.

# 3. MODELING HIGH INTEGRITY ASPECTS USING ROLE MODELS

Different integrity mechanisms can be modeled as aspects (e.g., auditing aspect, authentication aspect, data sensitivity aspect). To keep our paper concise, we focus on specifying an authentication aspect and an auditing aspect. Recall that authentication is the process of verifying that the identity claimed by a user is his true identity. Auditing is the process of recording the events in a system in an event log that is to be used for later analysis.

## 3.1 Modeling Authentication Aspect: Structural View

The authentication aspect from a structural perspective is shown in Figure 1. In this aspect, *Initiator* is intended to be played by classes whose instances will initiate the authentication request, *Target* is intended to be played by classes whose instances will receive this request, *IdentityRepository* is intended to be played by classes whose instances represent the repository of user identities, and *IdentityEntity* is intended to be played by classes representing user identities. Some of the features and constraints defined in the SRM are listed below:

- The behavioral role, *lookUp* in *IdentityRepository*, characterizes behaviors that look up an identity in the repository.
- Target has behavioral roles *verifyID* (verifies whether the identity is valid), *lockOut* (locks out the identity), and *startConvs* (initiates a conversation using the claimed identity of the *Initiator*).

## 3.2 Modeling Authentication Aspect: Interaction View

We use IRMs to describe the interaction pattern defined by an aspect. The interactions that take place when a user needs to authenticate itself are shown in Figure 2. Constraints on the ordering of messages are expressed in terms of *message sequence templates*. A particular ordering can be obtained by substituting sequence values (ordered list of natural numbers) for the template parameters that satisfy the parameter constraints. In Figure 2 there is one template parameter, *n*, which can be substituted by sequence values (e.g., 1.3.2).

The authentication process starts (message role *[[n]]*) with the client (an instance of a class that conforms to *Initiator*) sending a message containing

the claimed identity to the server (an instance of a class that conforms to *Target*). The server then sends a message to the identity repository to look up the claimed identity (message role *[[n]].1*). If the identity does not exist (message *[[n]].2A*), then the client is locked out (message role *[[n]].2A.1*) and receives an error message (message role *[[n]].2A.2*). Otherwise (message *[[n]].2B*), the server decides whether the claimed identity is the true identity (message *[[n]].2B.1*). If not (message role *[[n]].2B.2A*), the client is locked out (message role *[[n]].2B.2A.1*) and it receives an error message (message role *[[n]].2B.2A.2*). If the claimed identity is the true identity (message role *[[n]].2B.2B*), then control returns to the client but with a continue signal.

## 3.3 Modeling Auditing Aspects

We use SRMs to model the structural component of the auditing aspect (shown in Figure 3). In this aspect *Invoker* will be played by classes whose instances will invoke a method on a class that plays the role of *Invokee*. *Invokee* will be played by classes whose instances will invoke methods on a class playing the *SubInvokee* role as a result of executing the method invoked by an *Invoker* class. *Log* is intended to be played by classes whose instances will log the outcome of the *SubInvokee* methods. Some of the constraints defined by the SRM are:

- The multiplicities of the association between *Invoker* and *Invokee* are parameterized in the SRM. The constraints on the multiplicities are indicated by OCL constraints. The lower bound on the multiplicities is each equal to one.
- There is a *Log* that logs the outcome of *SubInvokee* methods invoked by *Invokee* as a result of executing *Method()*.

We describe the interaction component of the auditing aspect using the IRM shown in Figure 4. The *Invoker* sends a message (message playing the message role *[[n]]*) to the *Invokee*. The *Invokee* then executes the method, which involves invoking a method (represented by the role *otherOp*) of *SubInvokee* (message role *[[n]].[[p]]*). The *Invokee* then sends to the *Log* the identity of the *Invoker*, the method invoked (method that plays the role *otherOp*), and the outcome of the method (message role *[[n].[[q]]*, where the first element of $q$ is greater than the first element of $p$, i.e., a message playing the *[[n]].[[q]]* role occurs after the message playing the *[[n]].[[p]]* role).

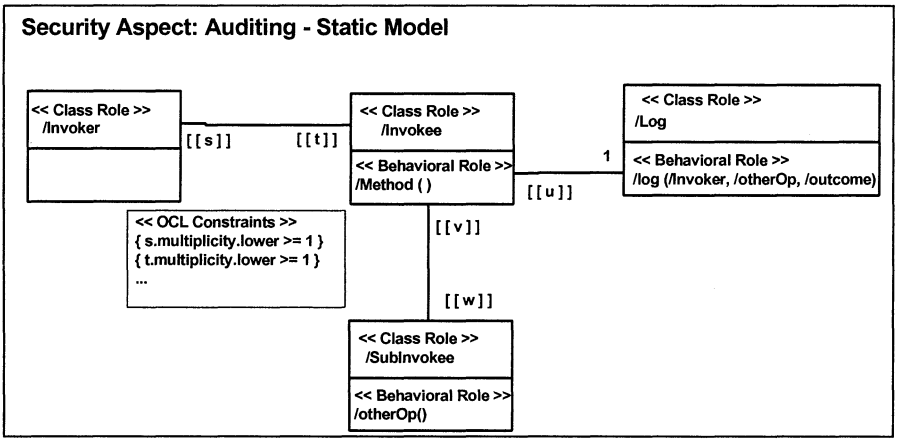


*Figure 3.* Static view of an auditing aspect.

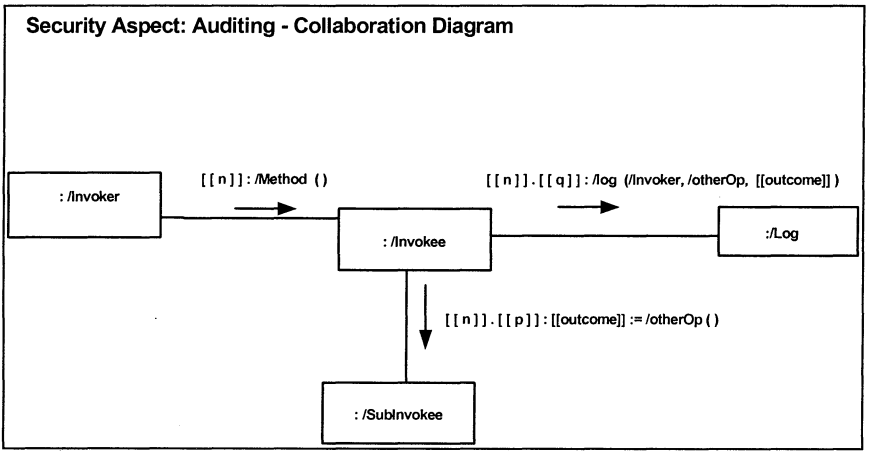


*Figure 4.* IRM for the auditing aspect.

## 4. WEAVING ASPECTS INTO A DESIGN MODEL

Weaving of an aspect into a primary model involves:

(1) *Mapping primary model elements to the roles they are intended to play*: Before the weaver can incorporate the aspect into a primary model the modeler must first indicate the parts of the model the aspect is to be woven into. The modeler can accomplish this by explicitly indicating the model elements that are intended to play the roles. Alternatively, the aspect can characterize the points into which it will be woven (as is done with pointcuts in AspectJ). In this paper the former approach is used. We are currently

developing support for the second approach. Note that not all model elements need be mapped to roles. Also, not all roles need be associated with a primary model element. Roles not associated with primary model elements indicate that new model elements must be created and added to the primary model as described later.

(2) *Merging roles with primary model elements*: Each model element that is mapped to a role has its properties matched with the properties contained in the aspect, and additional properties are generated from the role if deficiencies in the model element are found. For example, a class that is mapped to a class role that does not have the attributes or operations that play structural and behavioral roles defined in the mapped class role is extended with attributes and operations generated from the role.

(3) *Adding new elements to the primary model*: Each role that is not mapped to a model element is used to generate a model element that is then added to the model.

*(4) Deleting existing elements from the primary model*: If a model element is mapped to a <<delete>> role (a <<delete>> role indicates that conforming elements must not exist in the model), then the model element is removed from the primary model.

## 4.1 Weaving Aspect into Static Models using SRMs

We use a simple example of a user management system to illustrate our weaving technique. The Class Diagram shown in Figure 5 models the static structure of this system. The user management system consists of (i) *Managers* that direct actions on user information, (ii) *SystemMgmt* that carries out the action, (iii) *userList* that contains information about users and (iv) *userInfo* that contains information about individual users. We assume that there is a single class that plays the *SystemMgmt* and a single *userList*.

Figure 7 shows the result of weaving the static view of the *Authentication* aspect into the original *User Management* Class Diagram. *Manager* is mapped to *Initiator* and *SystemMgmt* is mapped to *Target*. *IdentityRepository* will be played by *userList* and *IdentityEntity* will be played by *userInfo*. Merging of roles to mapped primary model elements can result in modification to primary model elements. For example, the *SystemMgmt* class must be augmented to play the role of *Target* by adding operations of *verifyID()*, *lockOut()*, *startConvs()*. The *userList* class is augmented to play the *IdentityRepository* role by adding a *lookUp()* operation. The multiplicity of the association between *Manager* class

(playing the *Initiator* role) and *SystemMgmt* class (playing the *Target* role) must be changed from * to *1..**, that is, the aspect template *n* must be substituted with *1..**. Other multiplicity substitutions are shown in Figure 7. The aspect roles played by model elements are shown in the woven static diagram using stereotypes (see Figure 7).

## 4.2 Weaving Aspects into Interaction Models using IRMs

The collaboration diagram describing the *Add User* behavior is shown in Figure 6. The collaboration starts with the manager sending a request to add a user (message *1*). The system management, in response, sends a look up request to the user list (message *1.1*). After the user list has completed this request, control returns to the system management object. If the user already exists, an error message (*1.2B*) is sent to the manager. Otherwise (*1.2A*), two steps are performed in sequence – the add user request is generated (*1.2A.1*) and the operation complete signal is generated (*1.2A.2*).

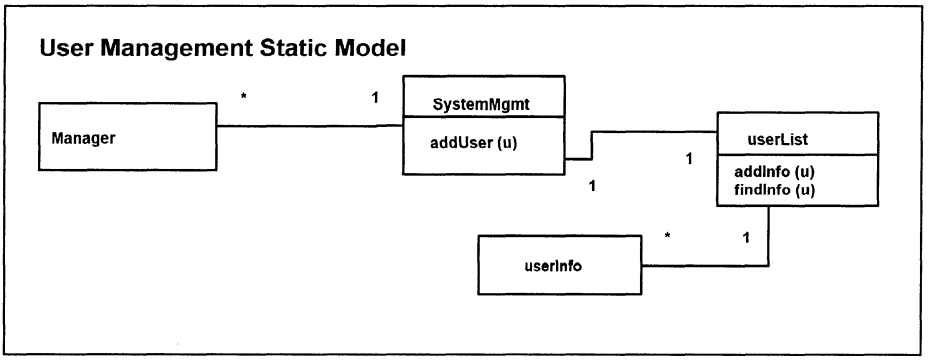


*Figure 5.* Design class diagram for a user management system.

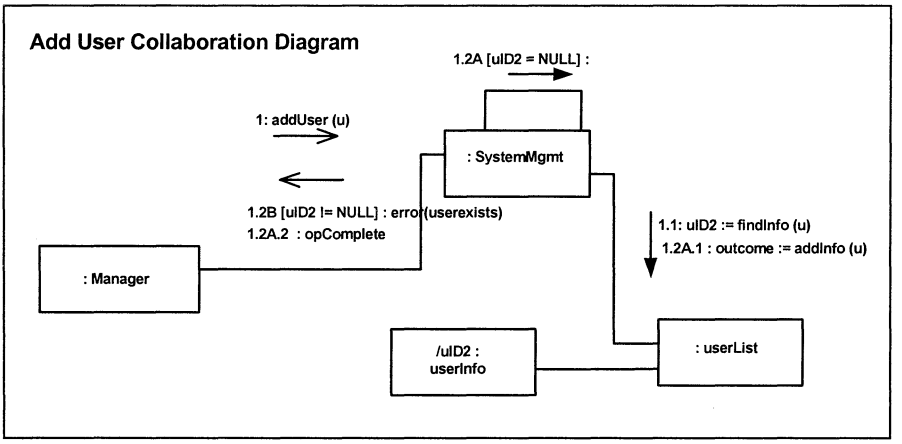


*Figure 6.* Collaboration diagram for adding a user to a distributed system.

Figure 8 shows the result of weaving the authentication aspect interaction view into this model of user management system. The weaving is accomplished as follows:

- The *id* parameter is substituted with *cid* and the *iden* parameter is substituted with *uID1*.

- The template parameter *n* in the IRM is set to *1* because authentication must be carried out before any operation is performed. Consequently, all the interactions shown in the primary model (shown in Figure 6) must be renumbered. We do this by adding one to the outermost sequence number; thus the interactions in Figure 8 are now represented by interactions *2*, *2.1*, *2.2A*, ... in the woven model (see boxed messages in Figure 8).

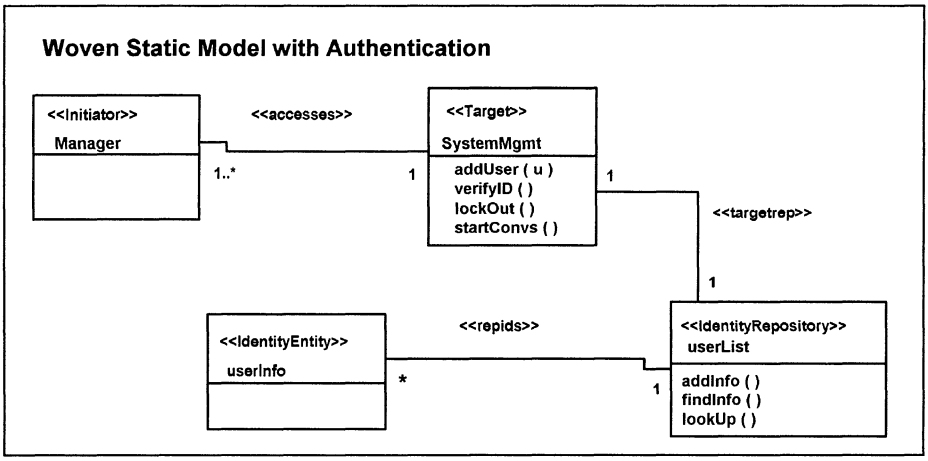


*Figure 7.* Static diagram of user management system with authentication woven into it.

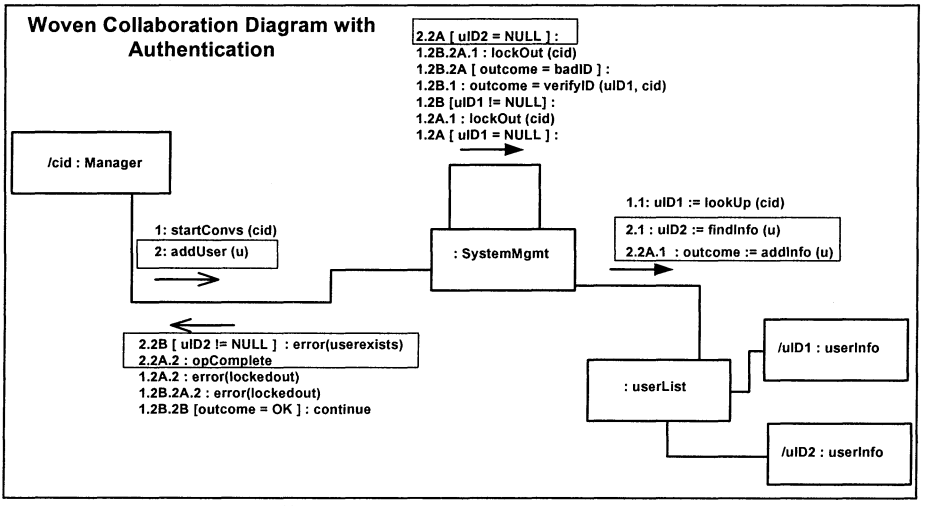


*Figure 8.* Collaboration diagram of adding a user with authentication woven into it.

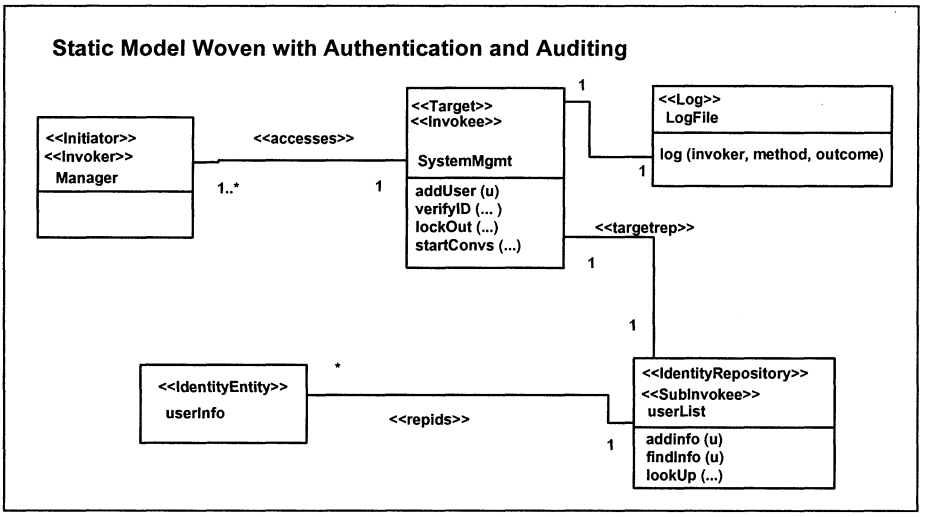


*Figure 9.* Static diagram with authentication and auditing woven into it.

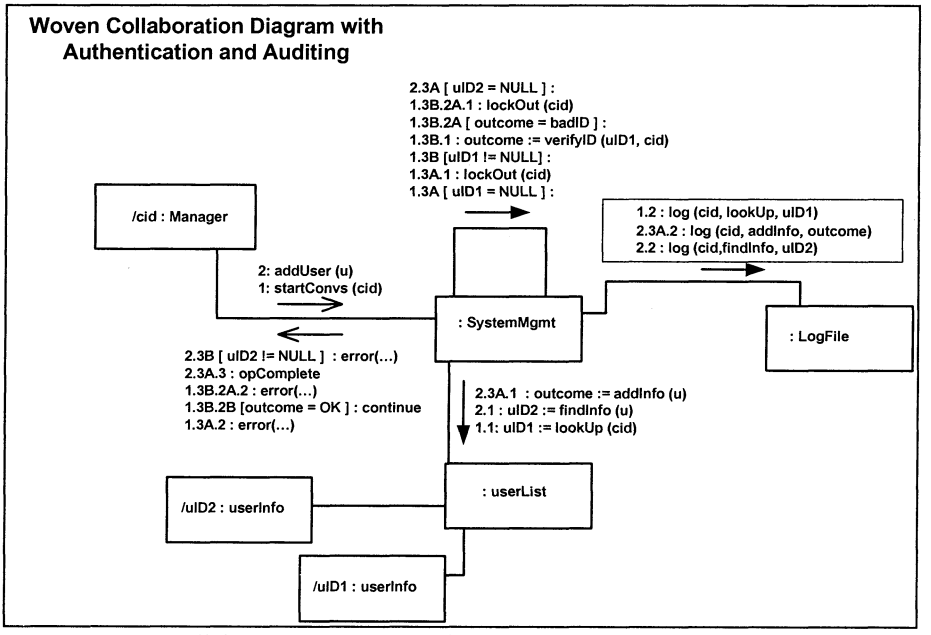


*Figure 10.* Collaboration diagram with authentication and auditing woven into it

## 4.3 Weaving Multiple Aspects

A system can be associated with multiple aspects – each aspect modeling an independent concern. It may appear that the order in which the aspects are woven is unimportant. Unfortunately, there are dependencies among the aspects and the order of weaving plays an important role. The weaving of multiple aspects with a primary model can proceed as follows:

(1) Analyze the dependencies between the aspects and decide on the order in which the aspects must be woven.
(2) Weave the first aspect with the primary model to get a woven model.
(3) Weave the next aspect (as dictated by the dependencies) with the previous woven model.
(4) Repeat step (3) until there are no more aspects to be woven.

We illustrate this approach using the authentication and the auditing aspect. There is a dependency between these aspects; auditing of authentication activities is desired. For this reason, the authentication aspect is woven before the auditing aspects. Figures 7 and 8 show the result of weaving the authentication aspect to the user management system.

Once the authentication aspect has been woven in, we are in a position to weave in the auditing aspect. Recall that the auditing aspects are shown in Figures 3 and 4. In the user management system, *Manager*, *SystemMgmt*, *userList* play the role of *Invoker*, *Invokee*, and *SubInvokee* respectively. The intent is to log the invocation of *userList* methods as a result of methods invoked on *SystemMgmt*. No existing model element plays the role of *Log*. The new model element, *LogFile*, is generated that will play the role of *Log*. The final static woven model is depicted in Figure 9.

The weaving of the collaboration aspect requires inserting log messages after the logged methods have been executed. The *startConvs()* method generates a number of method invocations and we are interested in logging *lookUp()*. In this case we have to add a log operation after execution of this method. This requires adding a new message (1.2), and renumbering subsequent messages (the former 1.2 becomes 1.3, and so on). In this case *n* and *p* are set to *1* (i.e., *[[n]].[[p]]* is *1.1*) and *q* is set to *2* (i.e., *[[n]].[[q]]* is *1.2*). For the *addUser()* method, methods *findInfo()* and *addInfo()* are to be logged. The log methods are given in the box shown in Figure 10.

The auditing aspect is concerned with logging methods invoked by a subject. The authentication aspect is concerned with authenticating a user. The authentication aspect invokes methods, such as, *lockOut()*. Thus, if the auditing aspect is woven before the authentication aspect, and the *Invoker* and *Invokee* roles are not associated with classes that will play the *Initiator* and *Target* roles, the methods corresponding to the authentication aspect will

not be logged. Thus, weaving the auditing aspect prior to the authentication aspect results in an incomplete woven model. Weaving the authentication aspect prior to the auditing aspect results in a correct woven model.

## 5. CONCLUSIONS

In this paper we propose a technique to model and integrate concerns into designs. The approach is aspect-oriented in that integrity concerns are modeled independently as aspects that can then be woven into models of essential functionality. In our approach, a listing of the different kinds of attacks/problems that are prevalent and the mechanisms required to protect against such attacks is used to identify needed aspects and the strategy for weaving them into a design. Thus, two levels of weaving rules are needed in our approach. The first is the weaving strategies that identify the integrity aspects needed for particular situations and that constrain how the aspects are woven into models of essential functionality (e.g. weaving order). The second level of weaving deals with the mechanics of the weaving process.

The weaving strategies are intended to be reusable forms of experiences that can be used to assess the threats to a particular system and propose techniques (i.e. a combination of mechanisms) to prevent or detect the related attacks. An interesting by-product of the weaving strategies is that we can change them (i.e., re-weave mechanisms), and see the impact on the system of these proposed changes. Note that the integrity provided by mechanisms in the model is only as good as the weaving strategies.

In this paper we have illustrated our approach on a very simple example. In future, we plan to apply this approach to real world examples. Real world examples typically will have a large number of aspects and many different kinds of weaving rules. Applying to real world examples will help us in formulating a methodology for designing complex high integrity systems using aspects.

Our experience indicates that flexible tool support for weaving will greatly enhance the practicality of our approach. Flexibility in the weaving process is necessary because the manner in which the aspects are woven is highly dependent on the form of source models and the type of aspects being woven. We are currently developing a prototype tool that will support flexible weaving, by providing users with a language for describing reusable weaving strategies and weaving procedures.

# REFERENCES

[1] G. J. Ahn and M. E. Shin 2001. Role-based authorization constraints specification using object constraint language. *Proceedings of the 10th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*: 157-162, Cambridge, MA, June.

[2] H. A. Ali 2001. A new model for monitoring intrusion based on Petri nets. *Information Management and Computer Security* **9**(4): 175-182.

[3] L. F. Andrade and J. L. Fiadeiro 2001. Coordination technologies for managing information system evolution. *Proceedings of the 13th Conference on Advanced Information Systems Engineering*. Interlaken, Switzerland, June.

[4] F. Bergenti and A. Poggi 1999. Promoting reuse in aspect-oriented languages by means of aspect views. Technical Report DII-CE-TR005-99, DII – Universita di Parma, Parma.

[5] L. Bergmans and M. Aksit 2001. Composing crosscutting concerns using composition filters. *Communications of the ACM* **44**(10), October: 51-57.

[6] M. T. Chan and L. F. Kwok 2001. Integrating security design into the software development process for e-commerce systems. *Information Management and Computer Security* 9(2-3): 112-122.

[7] S. Clarke and J. Murphy 1998. Developing a tool to support the application of aspect-oriented programming principles to the design phase. *Proceedings of the International Conference on Software Engineering*, Kyoto, Japan, April.

[8] P. T. Devanbu and S. Stubblebine 2000.Software Engineering for Security: a Roadmap. *Future of Software Engineering ICSE 2000 Special Volume.*

[9] Z. Diamadi and M. J. Fischer 2001. A simple game for the study of trust in distributed systems. *Wuhan University Journal of Natural Sciences* **6**(1-2): 72-82.

[10] J. L. Fiadeiro and A. Lopes 1999. Algebraic semantics of co-ordination or what is it in a signature? *Proceedings of the 7th International Conference on Algebraic Methodology and Software Technology*, Amazonia, Brasil, January.

[11] R. B. France, D. K. Kim, and E. Song 2002. Patterns as precise characterizatons of designs. Technical Report 02-101, Computer Science Department, Colorado State University.

[12] R. France, D. K. Kim, E. Song, and S. Ghosh 2001. Using roles to characterize model families. *Proceedings of the 10th OOPSLA Workshop on Behavioral Semantics: Back to the Basics,* Seattle, WA.

[13] R. France and G. Georg 2002. Modeling fault tolerant concerns using aspects. Technical Report 02-102, Computer Science Department, Colorado State University.

[14] G. Georg, I. Ray, and R. France 2002. Using aspects to design a secure system. *Proceedings of the 8th IEEE International Conference on Engineering of Complex Computer Systems*. Greenbelt, MD, December.

[15] L. Giuri and P. Iglio 1996. A role-based secure database design tool. *Proceedings of the12th Annual Computer Security Applications Conference*: 203-212.

[16] J. Gray, T. Bapty, S. Neema, and J. Tuck 2001. Handling crosscutting constraints in domain-specific modeling. *Communications of the ACM* 44(10), October: 87-93.

[17] R. Holbein, S. Teufel, and K. Bauknecht 1996. A formal security design approach for information exchange in organisations. *Proceedings of the 9th Annual IFIP TC11 Working Conference on Database Security*: 267-285, Rennselaerville, NY.

[18] J. Jurjens 2001. Towards development of secure systems using UMLsec. *Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering*: 187-200, Genova, Italy.

[19] J. Jurjens 2001. Modeling audit security for smart-card payment schemes with UML-SEC. *Proceedings of the IFIP TC11 16th International Conference on Information Security*: 93-107, Paris, France, June.

[20] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold 2001. Getting started with AspectJ. *Communications of the ACM* 44(10), October: 59-65.

[21] K. Lieberherr, D. Orleans, and J. Ovlinger. 2001. Aspect-oriented programming with adaptive methods. *Communications of the ACM* 44(10), October: 39-41.

[22] U. Nerurkar 2000. A strategy that's both practical and generic. *Dr. Dobb's Journal* **25**(11), November: 50- 56.

[23] P. Netinant, T. Elrad, and M. E. Fayad 2001. A layered approach to building open aspect-oriented systems. *Communications of the ACM* 44(10), October: 83-85.

[24] Object Management Group 2001. Unified Modeling Language Version 1.4. http://www.omg.org, September.

[25] H. Ossher and P. Tarr 2001. Using multidimensional separation of concerns to (re)shape evolving software. *Communications of the ACM* 44(10), October:43-50.

[26] J. A. D. Pace and M. R. Campo 2001. Analyzing the role of aspects in software design. *Communications of the ACM* 44(10), October.

[27] C. P. Pfleegler 1997. *Security in Computing, 2nd Edition*. Prentice-Hall.

[28] A. R. Silva 1999. Separation and composition of overlapping and interacting concerns. *Proceedings of the 1st Workshop on Multi-Dimensional Separation of Concerns in Object-Oriented Systems*. Denver, CO, November.

[29] G. T. Sullivan 2001. Aspect-oriented programming using reflection and metaobject protocols. *Communications of the ACM* 44(10), October: 95-97.

[30] D. Trcek 2000. Security policy conceptual modeling and formalization for networked information systems. *Computer Communications* **23**(17): 1716-1723.

[31] J. Warmer and A. Kleppe 1999. *The Object Constraint Language: Precise Modeling with UML*. Addison Wesley.

[32] J. J. Whitmore 2001. A method for designing secure solutions. *IBM Systems Journal* 40(3): 747-768.