Rational Software Architect
June 2004

Rational. software

Modeling Security Concerns in Service-Oriented Architecture

Abstract
Many enterprises are implementing service-oriented architecture (SOA) using Web services, and are designing those services according to the principles of Model Driven Architecture (MDA). Because the UML used to express MDA lacks model elements for indicating the security needs of business processes, system architects are forced either to ignore security concerns in their models, or to indicate their intentions in ways that are implementation-specific. This paper proposes a candidate profile for UML that presents security-related intent elements as stereotypes that business users and software architects can apply to UML elements when working with business stakeholders to capture business requirements. Using a profile such as the one proposed here would allow architects to specify the business intent of security in their designs without violating the MDA prohibition against implementation-specific details in high-level, behavioral models.

Simon Johnston
Architect
IBM Software Group

Contents

Introduction

A service-oriented architecture (SOA) is a way of designing software to provide services to applications, or to other services, through published and discoverable interfaces. Each service provides a discrete chunk of business functionality through a loosely coupled (often asynchronous), message-based communication model. System architects who take advantage of SOA can incorporate one or more services into their applications as components.

Much of the software industry's focus so far has been on the underlying technology for implementing Web services and their interactions. Insufficient attention has been given to the techniques and tools required for architecting enterprise-scale software solutions using Web services. The design of a high-quality software solution, like any other complex structure, requires early architectural decisions supported by well-understood design techniques, structural patterns, and styles. These patterns address common service issues such as scalability, reliability, and security.[1]

Business stakeholders depend on the IT organization to provide solutions to their business requirements. For both financial and market-driven reasons, stakeholders want to shorten the investment in time and money it takes to deliver IT solutions. They also want to increase the value they derive from IT solutions by maximizing the requirements coverage each software project provides. Because so many of those projects today involve Web services, it is very important that we have better tools and techniques for the rapid and successful implementation of those business requirements using SOA. We consider modeling to be especially important because of its ability to separate concerns[2] and present a unified view of those concerns. Security in service implementations is a major concern because many applications operate across organizational boundaries. The purpose of this paper is to provide a set of primitive modeling elements that allow the business stakeholders to specify the intent of security within the requirements process.

Architectural versus Implementation Models

As IT professionals rush to deliver applications using Web services, they often find themselves in the position of coming up to speed on an architectural model (SOA) and an implementation model (Web services) simultaneously. As one might expect under the circumstances, the distinctions between the model and the implementation sometimes get lost. This paper assumes the use of a Model Driven Architecture[3] approach that carefully avoids commingling the platform-independent model of an application's architecture and behavior with the technologies and platforms used to implement that modeled behavior. System architects employ either domain-specific languages or profiles for Unified Modeling Language (UML)[4] to model the concerns of the service domain. The principles that require architects to keep platform and language concerns out of this model also require them to keep implementation-specific security concerns out. For example, a model that includes abstract notions of services and messages should not also include details of how messages can use public-key encryption and certificates to implement service authentication and message signatures, because doing so violates a very fundamental principle (the need to separate concerns) by introducing the details of a particular technical implementation into the platform-independent model. On the other hand it is not possible to treat security as an afterthought. Security implementations are complex; they can have serious impacts on performance, and they can place additional requirements on the IT infrastructure supporting the services. It is therefore in everyone's best interests to model security concerns as carefully as any other concerns.

By separating concerns, the IT organization can successfully engage the business stakeholders in understanding and describing the business need to maximize requirements coverage. We intend to show how to specify security intents in these high-level models while isolating behavioral concerns from implementation and platform-specific ones, consistent with the principles of MDA.

Revisiting Security Concerns

The team that developed the RosettaNet[5] family of B2B standards raised issues that were similar to the ones we just discussed, and made a start toward addressing them. The idea was to present a simplified set of choices to the business architects — the primary stakeholders from whom the RosettaNet team gathered data and processing requirements. These business architects were not versed in the technical details of the security concerns, but they were able to distinguish data that needed to be passed in a secure manner from data that could be sent without security measures.

However, one problem with this approach was that a simplified terminology was vital. As soon as the terms became complex or opaque, the business stakeholder would request every available kind of security, just to be safe, and that resulted in sub-optimal designs. It was this behavior on the part of stakeholders that led the architecture group to set out simple guidelines on the specification of such constraints as well as descriptions that the user could understand. Doing so gave stakeholders the insight they needed to make informed cost/benefit trade-off decisions. For example, the RosettaNet team used examples to help the business user understand when the cost of encrypting data was far greater than the value of the data being protected.

Generalization of Security Issues

There are many texts on general software security issues, and many more on particular security implementations and technologies. However, we want to be able to address the underlying intent that drives security-related technical implementations. Specifically, we would like to be able to specify a set of descriptive primitive intents that are easy to understand, and that can be used to identify particular technical implementations.

What underlying issues and concerns does the term "security" include? Let us take a common example: withdrawing cash from an ATM machine. First, when I walk up to an ATM machine, I am asked to provide two things: my ATM card (which acts as formal identification) and a personal identification number (PIN), which is a "shared secret" known to me and my bank, but to no one else. The ATM can now take the identification information and PIN and ask my bank whether the person standing in front of it can be assumed, with an acceptable level of confidence, to be the account holder. If the issuing bank approves the supplied details, it sends back to the ATM a set of security credentials that provide additional information on the account holder. Based on this account-specific information, the ATM then displays the list of actions I am authorized, as the account holder, to perform — actions that might include "withdrawal", "deposit," and so on. Note that this set of options actually represents the intersection of two sets:

- The set of all operations this specific ATM is capable of performing
- The set of all operations the issuing bank certifies that I am eligible to perform

The credentials sent by the bank typically include a limit on the amount of cash I can withdraw in any single transaction-a limit that the ATM itself can enforce. The ATM system architect must also provide an audit trail by logging all the information flowing to and from the ATM.

How is this communication between bank and ATM accomplished? How can the bank trust the information it gets from the ATM and vice versa? Technical details like these, related to security protocols, data encryption and so on, are both important and interesting — but these are precisely the kind of details that are outside the scope of any high-level behavioral model.

The ATM example illustrates three categories of security concerns or domains:

- Who are you? (Identification, Authentication)
- What can you do? (Authorization)
- What can you, and others, see? (Privacy)

A fourth domain is less obvious, but it is intertwined with the other three domains:

- What happened? (Audit)

The audit domain often tends to be an afterthought in the development of many IT applications. On the other hand, in some business areas, such as core security, and in applications such as EDI (Electronic Data Interchange), where regulatory concerns require significant audit capabilities, the audit function is an explicit and important security concern. Our approach treats auditing as an implicit intent; consequently, the primitives we introduce all imply an audit trail of their detailed behavior. So, for example, when we mark that party A requires that before it can collaborate with party B, it must authenticate party B, this implies that that authentication request and response with the date/time as well as other implementation details must all be audited.

Figure 1 demonstrates the dependencies between these domains. For example, it is not possible to implement authorization without authentication. On the other hand both authorization and authentication rely on auditing, not for implementation but to ensure that any exceptions are captured for analysis and non-repudiation.

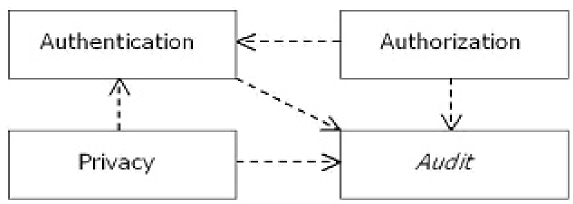

Figure 1: Dependencies between Security Domains

This paper goes on to address the primitive intents that are common in these domains and then describes how such primitives, once introduced into a model, can drive implementations in any given technology.

Who Are You?

This domain is primarily concerned with the identification of one or more parties to a communication or collaboration. We can actually separate this into two distinct concerns: the static notion of identification and the dynamic notion of authentication. In the ATM example, the ATM card is the static, bank-issued identification, whereas the card/PIN pair allows the ATM to authenticate me dynamically as the account holder. It is far more important to model the notion of authentication than identification; in general, identification becomes much more entwined in the technical details of its implementation than does authentication. So, for example, we can state very simply that in a particular collaboration between a set of parties there is a need for authentication between those parties. This can be done explicitly or implicitly; for example, another notion, that of trust, can be used to describe trust zones. Parties within the same trust zone can assume they do not need to authenticate each other, whereas for all sensitive communication across trust boundaries, authentication is required.

While the trust zone is a useful concept, it does not meet all needs. Trust zones are often hierarchical, and some communication with external parties can be accomplished without needing to authenticate the incoming party. A business might see itself as a single trust zone in the sense that no one except employees can access network resources inside the firewall. However, a single-zone picture like this will be misleading if, as often happens, there is also a trust zone separating the ERP (Enterprise Resource Planning) and CRM (Customer Relationship Management) applications, because each of these implements and maintains its own application-level security. The business might also see itself as a member of an external trust zone through its membership of a trading extranet. In this regard, we propose that trust and authentication are explicit, yet overlapping, intents that can be applied to a collaboration model.

What Can You Do?

This domain is primarily responsible for ensuring that, once we know who you are, we can restrict your options to only those operations you are authorized to perform. This requires the ability to perform authentication, because we must know who you are before we can decide what you are able to do. The business stakeholder wishes to identify those functions that need to be made secure, in the sense that we know who might perform them and we know (through audit) when they were performed. There are of course functions that do not require authorization, either because we want them to be accessible to everyone, or because, for performance reasons, we have identified a trust zone within which services can safely assume the requester's right to access them.

This trust zone notion becomes important when we consider performance as another, sometimes competing, concern, because there is a cost associated with implementing security, and sometimes that cost is quite high. Consider a function to return the number of outstanding orders for a customer — a very simple query from a database. If we ask for authentication, authorization, and privacy (see below), we are forced accept significant costs:

- We have to ask the requester to provide credentials
- We have to check those credentials, probably with a remote service
- We have to encrypt the returned information

If we know that the requester and provider are both services of the same application, we can identify them as a single trust zone, dispense with the overhead, and reap the benefit in increased performance. From an architectural standpoint it is also important to understand all the communication that goes on between trust zones. Such communication should be minimized and controlled as much as possible because it represents the most likely point of failure (or attack) in the overall security implementation.

In terms of implementation, there are two primary approaches to authorization that are interesting to note here.

- **Authorization of Individual Parties** — Every party can be assigned an explicit set of access rights to functions (although to optimize the process, we might agree to treat the absence of an explicit access right as either and implicit approval or disapproval of that access right).
- **Authorization through Roles** — Various roles can be created for each application, and access rights assigned as described above to those roles instead of to individual parties. When each party is authenticated, the credentials supplied for the party include the party's role, which then determines whether the party is authorized to access a particular function.

The important point to note here is that we always want to exclude details such as these from our high-level behavioral models. Our intent at the high-level modeling stage is simply to note, for example, that a given function requires authorization before it can be performed, without detailing how that authorization will be accomplished.

What Can You, and Others, See?

The concern of the privacy domain is to ensure that you see only the information that is you are authorized to see, and that other parties do not see information they are not authorized to see. Businesses both consume and generate large amounts of data, which is stored, manipulated and passed around to support the running of the business. We have to ensure that information that is sensitive in nature is protected and that we provide a way to know who is requesting or providing information and services. That is to say, we need to know not simply which service was the requester or provider, but which authenticated end user was a party to the transaction through that service.

There are two distinctly different intents associated with the privacy domain:

- The intent to sign a message or document (potentially using multiple signatures) identifying the end user(s) or service(s) that created the message or document. This intent, which relies on a number of common standards for digital signatures, is used in B2B commerce, government commerce, and even more widely in corporate email communications. A document may have a number of signatures, for example a supply requisition may be signed by the originator, their manager (approver) and finally the purchasing department when the order is placed; all of these signatures travel with the document.
- The intent to ensure the privacy of a message, either by sending it over a secure medium or by encrypting or otherwise securing the content. This intent is probably the area with the most variability in the choices for implementation. For example, in assuming a digital message transferred between services we can consider that the message itself is encrypted by the sender and then sent over an insecure channel, that the message is sent as plain text over a secure transport such as HTTPS or TLS (both of which in fact provide encryption as a part of the service, but outside of the sender's control) or even that the message contains an identifier for a document sent by some other secure means (such as printed on copy-proof paper and sent by courier).

Another concern that is often cited is that of data integrity — the need to ensure that the contents of a message or document cannot be altered en route between the various parties to the transaction. A message or document that has data integrity is said to be tamperproof. The expectation is that if a message is noted as private it should be tamperproof as well; however, it should also be possible to indicate that a message is simply tamperproof without requiring a privacy solution.

Ultimately, the implementer's role is to provide a solution that meets the requirements of data privacy in common with guidelines laid down by the business. For example, it would not be acceptable for an e-commerce Web site to use a courier to transmit each customer's credit card number to the bank for verification, whereas couriers might be an excellent choice for delivering top-secret government documents.

It Wasn't Me!

Another area of concern is the notion of non-repudiation of origin and content, a term you will find in many EDI documents. Non-repudiation is a grand title for the notion that at some point in the future one of the parties to a transaction might deny having completed the transaction. Alternatively, one of the parties to a transaction might acknowledge that the transaction took place, but dispute a specific detail of that transaction. For example, having purchased 100,000 shares of a stock that subsequently lost value, party A might attempt to claim that the transaction was for only 100 shares. In many industries and geographies there are legislative regulations that require the keeping of records for lengthy periods to arbitrate such disputes.

In this regard the auditing concern we introduced earlier should provide the necessary capabilities to store messages. While auditing alone does not necessarily suffice for the purposes of non-repudiation, auditing of the message exchange (proof of content) along with authentication (proof of origin) generally does provide the required level of proof.

Applying the Primitives to a Model

The following is an example of how our candidate profile, described fully in the "Profile Details" section of this paper, might be used to denote security intents in a simple, high-level model of a document exchange between two parties. The table below summarizes the intent elements used in the example.

Intent	***Comments***
audit	Used to denote that the specified communication is to be audited. One can assume that the audit will contain both the authenticated identity of all parties as well as any data communicated between the parties.
authenticate	Used to denote a party that has to be authenticated in the scope of a given collaboration.
authorize	Used to denote that a communication between two parties must ensure that the requester is authorized to perform the request.
private	Used to denote that the marked information should be treated as private and that all reasonable effort (considering technical implications) should be made to ensure the data is both private (protected against unauthorized viewing) and tamperproof (guaranteed to arrive at its destination without modification).
signed	Used to denote that the marked information includes the digital signatures of parties related to the document.
tamperproof	Used to denote that the data transferred between parties must be guaranteed to reach the recipient in the same form and with the same content and meaning as when it left the sender.

Intent	Comments
trusted	Used to denote a set of parties in a collaboration that participate in an explicit trust zone.

The following is a snippet of a UML activity model that demonstrates the exchange of purchase orders between a buyer and seller. Notice that the intent elements in this example are realized as UML stereotypes applied to the general model elements.

Figure 2 illustrates three security intents for this collaboration:

- The buyer needs to be authenticated by the seller
- The "Accept PO" action requires authorization
- The purchase order object flow must be signed and in one case must also be tamperproof.

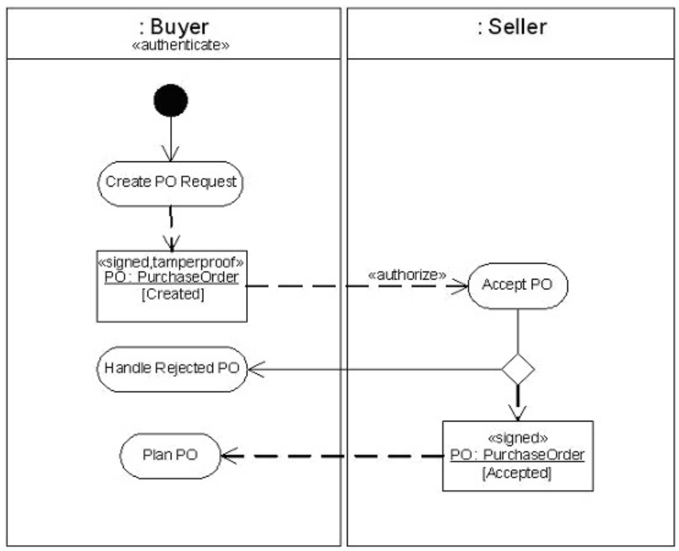

Figure 2: Security Intents in Purchase Transaction

It is clear that the model contains no additional technology concerns: no mention of service, end-points, interfaces, schemas, XML, and so forth. This is the high-level platform independent view that we recommend as a vehicle for eliciting security-related requirements from the business stakeholders.

It is worth noting that the profile presented in the "Profile Details" section of this paper assumes very little about the modeling tools or methods you might wish to employ in developing the platform-neutral model. For example, whereas Figure 2 uses an activity diagram to model business behavior, Figure 4 uses a sequence diagram, and it is also possible to use collaboration diagrams and even state machines to model behavior of business applications.

Example Mapping from Primitives to Implementation

The following examples demonstrate potential mappings from some of the primitives introduced above to particular technology designs. We will address actual implementation choices later in the paper. Again, we assume an MDA approach where the high-level models are transformed into implementation-specific models through the use of model-to-model transformations, which in this case are embodied in the patterns described below.

Protocols and Patterns

First, we need a way to express the technical implementation. In the UML, templated collaborations are used to represent patterns that can then be bound to particular model elements to expand them with additional details. In our case, we would need to develop patterns that represent technology-specific implementations for each of the intents in the model.

Figure 3 shows the pattern for authorization of a method using a trusted validation service. Notice that the IT group might well have a catalog of patterns to choose from for different implementations or potentially different characteristics such as performance. In the example, you can see that the pattern takes three parameters: the requester object, the method to be authorized, and the validation service to use. Figure 3 also demonstrates the use of our trust zone notation (the box that encloses the provider and validater objects), which visually brings to our attention the requirement that our validation service be trusted by the provider of the authorized method. This is important because the knowledge that a trust zone exists allows us to dispense with implementing security between two services that might interact frequently, in which case we would want to optimize them for performance.

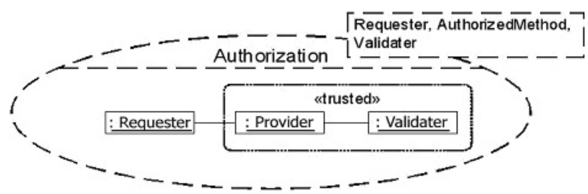

Figure 3: Pattern for Authorization

Within the pattern we can then describe the behavior of the implementation. Figure 4 shows how the provider calls the authenticator to validate the requester's credentials and, depending on the authenticator's reply, either executes the method or signals an authorization exception.

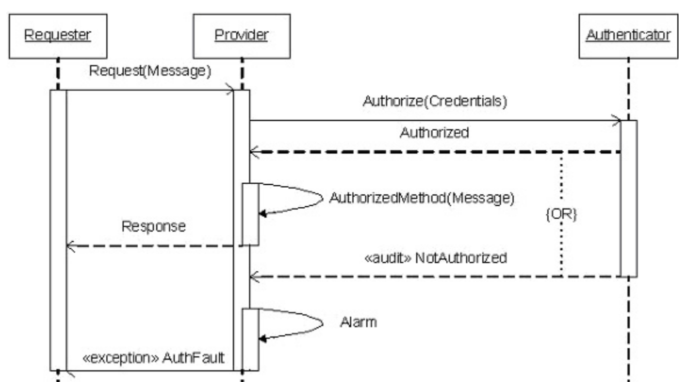

Figure 4: Implementation Behavior

This example makes use of a UML message sequence diagram that shows the sequence of events between all three of the parties defined in Figure 3. Note that additional stereotypes have been introduced in Figure 4, and in particular, that any message response from the validation service denoting a failed authorization is to be audited.

We mentioned earlier that most of the explicit intents imply an audit requirement. Yet in Figure 4, you can see that the NotAuthorized reply from the Authenticator is explicitly marked with the stereotype audit. The reason is that there is a business requirement (in addition to any implied security requirement) to audit this event.

Note that the stereotype exception is part of the core UML specification and not a part of our proposed profile. Note also that the constraint {OR} is used to model on a single diagram both possible outcomes of the authorization method.

To connect the pattern to our example activity model, we have to replace the parameters in the pattern with elements from our model, as shown in Figure 5. We create a binding that has the Buyer object as the requester, the Accept PO action as the guarded method, and a service called XWSKeySvr as the validater. This is creates an instance of the pattern in the model. We can then bind the same pattern to many other instances of messages such as "Accept PO" within our platform-neutral model.

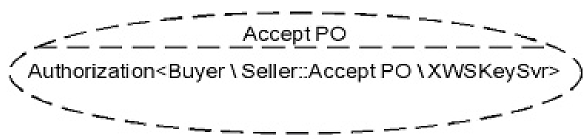

Figure 5: Binding the Pattern to the Model

This binding persists in the model, and therefore allows us at a later date to manipulate the bindings to create alternate implementations (for example, changing the validater from one service to another).

Implementation Choices

When developing the design patterns that are used to drive the implementation of intents, such as authorization or privacy, there are a number of design approaches and platform-specific technical solutions. For example, we have already discussed the notion of separating parties from roles in implementing authorization. In fact this is a very common approach and can be seen in both the J2EE and Microsoft .NET middleware. It is not the focus of this paper to go into detail on these design decisions and patterns, but we do expect that specific SOA patterns will appear, either as variations of common patterns such as the Gang-of-Four[6] or as entirely new patterns specific to the technical constraints and realities of a service-oriented application infrastructure.

Standards will also play a large part in the adoption of SOA, and in shaping exactly what SOA becomes. Base standards such as those from the World Wide Web Consortium (W3C) or the Organization for the Advancement of Structured Information Standards (OASIS) will form the foundation. Industry standards such as those from RosettaNet will provide the content and processes that will tie businesses together both internally and with their partners. However, we should be careful not to paint too rosy a picture. Many of these standards are relatively new, there are many commercial and academic interests represented in the development of them, and even an organization set up to administer standards to organize the standards (WS-I).

The following table represents a snapshot of the current specifications relevant to security in the web services space. The issue with these specifications, apart from the web of relationship between these specifications and between them and the base W3C and OASIS XML specifications, is that they are complex and are of course subject to change.

XML	***Messaging***	***Security***	***Related***
XML	SOAP	XML Encryption	WS-Policy
XML Namespaces	MTOP	WS-Security	WS-PolicyAssertions
XML InfoSet	WS-Addressing	WS-SecureConversation	WS-PolicyAttachment
XInclude	WS-Routing	WS-Trust	WS-SecurityPolicy
XPath		WS-Federation	XML Query
		Active Requestor Profile	
		Passive Requestor Profile	
		Web Services Security Kerberos Binding	
		Web Services Security Kerberos Binding	

We feel service-oriented architecture and the Web service implementation to be a great advantage to many IT organizations in the integration opportunities it opens up. We also feel strongly that attempts to re-architect existing applications using Web services should be carefully modeled and thoroughly understood, and that all significant concerns should be addressed separately in collaboration with the business stakeholders.

Profile Details

This section describes a candidate profile for UML that presents the intent elements as stereotypes that can be applied to UML elements in capturing the requirements of the business stakeholders.

stereotype audit

Meta Classes

ActivityNode, Message

Description

Used to denote that the specified communication is to be audited. One can assume that the audit will contain both the authenticated identity of all parties as well as any data being communicated between the parties.

Auditing is implicit in any communication stereotyped "authorize", it is also explicit in the implementation of authentication as well as in the exception handling for signed and private data.

In applying to an ActivityNode you can annotate actions, structured activities and control nodes (decisions for example) in an Activity diagram. In applying to a Message on an interaction you can annotate the messages sent between the represented model elements.

Properties

None.

Notation

No required notation.

stereotype authenticate

Meta Classes

ActivityPartition, Lifeline

Description

Used to denote a party that has to be authenticated in the scope of a given collaboration. The stereotype is applied to elements in a behavioral model that represent instances of a party in a collaboration.

In applying to either an ActivityPartition (for Activity diagrams) or Lifeline (for Interaction diagrams) you may annotate the represented model elements.

Properties

None.

Notation

No required notation.

stereotype authorize

Meta Classes

ActivityNode, Message

Description

Used to denote that a communication between two parties must ensure that the requester is authorized to perform the request. This stereotype is applied to messages and flows in behavioral models to denote that the behavior being invoked is guarded by an authentication check.

Properties

None.

Notation

No required notation.

stereotype private

Meta Classes

ObjectNode, Class

Description

Used to denote that the data transferred in a communication should be treated as private and that all reasonable efforts (considering technical implications) should be made to ensure the data is tamperproof and secure.

Do not apply the stereotypes tamperproof and private to the same element because private implies tamperproof. That is, applying the stereotype private specifies that the data will be both private (protected against unauthorized viewing) and tamperproof (guaranteed to arrive at its destination without modification). By contrast, applying the stereotype tamperproof specifies that the data will be guaranteed to arrive at its destination without modification without being protected against unauthorized viewing.

When applied to a Class (or derived UML element), it denotes that this element is signed whenever it appears in a behavioral model. The stereotype can also be applied to an instance in a behavioral model to denote that in this particular case the element is to be treated specially.

Properties

None.

Notation

No required notation.

stereotype signed

Meta Classes

ObjectNode, Class

Description

Used to denote that the data transferred in a communication includes some notion of a signature identifying a party. Note it is important that the element need not be signed by the party communicating, and that multiple signatures can be included. Note that the profile does not specify whose signature is required, nor does it specify how many signatures are required.

When applied to a Class (or derived UML element), it denotes that this element is signed whenever it appears in a behavioral model. The stereotype can also be applied to an instance in a behavioral model and denote that in this particular case the element is to be treated specially.

Properties

None.

Notation

No required notation.

stereotype tamperproof

Meta Classes

ObjectNode, Class

Description

Used to denote that the data transferred between parties must be guaranteed to reach the recipient in the same form and with the same content and meaning as when it left the sender.

Do not apply the stereotypes tamperproof and private to the same element because private implies tamperproof. That is, applying the stereotype tamperproof specifies that the data will be guaranteed to arrive at its destination without modification without being protected against unauthorized viewing. By contrast, applying the stereotype private specifies that the data will be both private (protected against unauthorized viewing) and tamperproof (guaranteed to arrive at its destination without modification).

Properties

None.

Notation

No required notation.

stereotype trusted

Meta Classes

ConnectableElement

Description

Used to denote a set of parties in a collaboration that participate in an explicit trust zone.

The ConnectableElement in this case is intended to be the set of elements representing the roles in a collaboration (as shown in figure 6).

Properties

Name	*Type*	*Comments*
Zone	String	The name of the zone and the participants in a zone become the set of elements in a given model with the same zone name.

Notation

It is useful to be able to denote, graphically, a zone boundary specifically in a collaboration diagram. An example of this is shown in the specification of the Authorization pattern. As shown in Figure 6, the trust zone is denoted by a dotted boundary drawn about its participants.

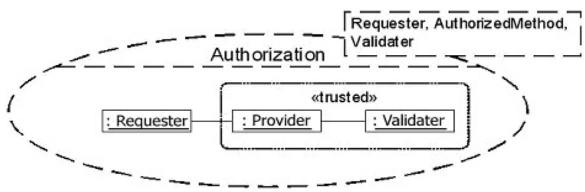

Figure 6: Denoting a Trust Zone Boundary

References

[1] Brown, A., Johnston, S., and Kelly, K, U*sing Service-Oriented Architecture and Component-Based Development to Build Web Service Applications*, Rational Software White Paper

[2] Lopes, C.V. and Hursch, W.L., *Separation of Concerns*, Tech Report of College of Computer Science, Northeaster University, Boston, MA, Feb. 24, 1995.

[3] OMG, MDA, *An Introduction*, OMG

[4] OMG, *UML 2.0 Superstructure Specification*, OMG

[5] RosettaNet Consortium [www.rosettanet.org]

[6] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison Wesley

Note that this paper is a reprint of an original article published by the IBM developerWorks web site.

IBM Corporation
Software Group
Route 100
Somers, NY 10589

Produced in the United States of America

March 2005.