Towards An Architectural Treatment of Software Security: A Connector-Centric Approach

Jie Ren, Richard Taylor, Paul Dourish, David Redmiles
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
1-949-8242776
{jie, taylor, jpd, redmiles}@ics.uci.edu

ABSTRACT
Security is a very important concern for software architecture and software components. Previous modeling approaches provide insufficient support for an in-depth treatment of security. This paper argues for a more comprehensive treatment based on software connectors. Connectors provide a suitable vehicle to model, capture, and enforce security. Our approach models security principal, privilege, trust, and context of architectural constituents. Extending our existing architecture description language and support tools, our approach can facilitate describing the security characteristics of an architecture generating enabling infrastructure, and monitoring run-time conformance. Initial results of applying this approach are illustrated through a case study. The contribution of this research is a deeper and more comprehensive treatment of architectural security through software connectors.

Categories and Subject Descriptors
D.2.2 [**Design Tools and Techniques**]: Modules and interfaces.

General Terms
Design, Security, Languages

Keywords
Software architecture, secure software connector, security

1. INTRODUCTION

With rapidly advancing hardware technologies and ubiquitous use of computerized applications, modern software is facing challenges that it has not seen before. More and more software is built from existing components. These components may come from different sources. This complicates analysis and composition, even if a dominant decomposition mechanism is available. Additionally more and more software is running in a networked environment. These network connections open possibilities for malicious attacks that were not possible in the past. These situations raise new challenges on how we develop secure software.

Conference'04, Month 1–2, 2004, City, State, Country.

Traditional security research has been focusing on how to provide assurance on confidentiality, integrity, and availability. However, with the exception of mobile code protection mechanisms, the focus of this research is not how to develop secure software that is made of components from different sources. Previous research provides necessary infrastructures, but a higher level perspective on how to utilize them to describe and enforce security, especially for componentized software, has not received sufficient attention from research communities so far.

Take a popular web server, Microsoft Internet Information Server (IIS), as an example. The web server was first introduced in 1995. It has gone through several version changes during the following years, reaching Version 5.1 in 2001. Along this course, it was the source of several vulnerabilities, some of which were high profile and have caused serious damages [3]. A major architectural change was introduced in 2003 for its Version 6.0. This version is much safer than previous versions, due to these architectural changes [29]. No major security technologies were introduced with this version. Only existing technologies were rearchitected for better security. This rearchitecting effort suggests that more disciplined approaches to utilize existing technologies can significantly improve the security of a complex, componentized, and networked software system.

Component-based software engineering and software architecture provide the necessary higher-level perspective. Security is an emergent property, so it is insufficient for a component to be secure. For the whole system to be secure, all relevant components must collaborate to ensure the security of the system. An architecture model guides the comprehensive development of security. Such high-level modeling enables designers to locate potential vulnerabilities and install appropriate countermeasures. It ensures that security will not be compromised by any single component and enables secure interactions between components. Architecture also allows selecting the most secure alternatives based on existing components and supports continuous refinement for further development.

Facing the new challenges of security for networked componentized software and given the base provided by existing software architecture research, we propose a software architecture technology that focuses on security. Our effort explores an approach related to aspect-oriented software architecture [7]. It provides a comprehensive treatment of security at the architecture level, using connectors as the central construct.

Section 2 of this paper surveys related work. Section 3 outlines our approach, introducing the modeling concepts and techniques we are exploring. Section 4 illustrates the approach through a component-based secure file sharing application that we develop. Section 5 summarizes initial results of our research and outlines future work.

2. RELATED WORK

Since our work is focused on semantically rich secure connectors, this section first surveys existing research on connector-based software architectures. Security has also been treated as an aspect by aspect-oriented modeling approaches at a higher design level. Thus, this section also surveys related efforts in this direction.

2.1. Architectural Connectors

Architecture Description Languages (ADLs) provide the foundation for architectural description and reasoning [21]. Most existing ADLs support descriptions of structural issues, such as components, connectors, and configurations. Several ADLs also support descriptions of behaviors [1, 20]. The description of behaviors is either centered around components, extending the standard "providing" and "requiring" interfaces, or is attached to connectors, if the language supports connectors as first class citizens [1]. These formalisms enable reasoning about behaviors, such as avoidance and detection of deadlock. Some early efforts have been invested on modeling and checking security-related behaviors, such as access control [23], encryption, and decryption [4].

Among the numerous ADLs proposed, some do not support connectors as first class citizens [10, 20]. Interactions between components are modeled through component specifications in these modeling formalisms. This choice is in accordance with component-based software engineering, where every entity is a component and interactions between components are captured in component interfaces. A component has a "provided" interface that lists the functionality this component provides. It also has a "required" interface that enumerates the functionalities it needs in providing its functionality. Interactions between components are modeled by matching a component's "required" interface to other components' "provided" interfaces.

Embedding interaction semantics within components has its appeal for component-based software engineering, where components are the central units for assembly and deployment. However, such a lack of first class connectors does not give the important communication issue the status it deserves. This lack blurs and complicates component descriptions, which makes components less reusable in contexts that require different interaction paradigms [9]. It also hinders capturing design rationales and reusing implementations of communication mechanisms, which is made possible by standalone connectors [12]. We believe a first class connector that explicitly captures communication mechanisms provides a necessary design abstraction.

Several efforts are focused on understanding and developing connectors in the context of ADLs. A taxonomy of connectors is proposed in [22], where connectors are classified by services (communication, coordination, conversion, facilitation) and types (procedure call, event, data access, linkage, stream, arbitrator, adaptor, and distributor). Techniques to transform an existing connector to a new connector [26] and to compose high-order connectors from existing connectors [19] are also proposed.

However, these efforts are not completely satisfactory. They suffer from the fact that they are general techniques. All of them aim at providing general constructs and techniques to suit a wide array of software systems, which leave them ignoring specific needs that arise from different application properties. For example, both the connector transformation technique [26] and the connector composition technique [19] have been applied to design secure applications, but the treatment of security does not address the more comprehensive security requirements as understood by security practitioners. Those requirements have richer semantics. These semantics raise challenges, because the general techniques must handle them in a semantically compatible way instead of just decomposing the challenges into semantically neutral "assembly languages." These semantics also provide opportunities, because they supply new contexts and information that can be leveraged. Such extra constraints are especially beneficial to the application of formal techniques, because these additional conditions could reduce the possible state space and lower the decidability and computational cost.

It is our position that a deeper treatment of security in the connector technology is needed for a comprehensive solution to the important software security problem. Such a treatment should handle and leverage the richer semantics provided by specific security properties, such as various encryption, authentication, and authorization schemes, instead of equating these security features with opaque abstract functions.

2.2. Aspect-related approaches

Recently, aspect-oriented programming concepts and techniques have been applied to high-level software design. These aspect-oriented design methods model security as an aspect.

2.2.1. UML-based Security Modeling

UML is a standard design modeling language. There have been several UML-based approaches for modeling security. UMLsec [15] and SecureUML [18] are two UML profiles for developing secure software They use standard UML extension mechanisms (constraints, tagged values, and stereotypes) to describe security properties.

Aspect-Oriented Modeling [24] models access control as an aspect. The modeling technique uses template UML static and collaboration diagrams to describe the aspect. The template is instantiated when the security aspect is combined with the primary functional model. This process is similar to the weaving process of aspect-oriented programming. The work described in [16] uses concern diagram as a vehicle to support general architectural aspects. It collects relevant UML modeling elements into UML package diagrams.

2.2.2. Other Aspect-Oriented Approaches

As pointed out in [14], aspect-oriented concepts can be utilized with modeling mechanisms that have no executable

semantics. However, such semantics do facilitate the use of aspect technology. Petri Net is proposed as a formalism to model software architecture [10], and [27] extends the work by using the executable semantics of Petri Nets as the basis for modeling access control checks during normal transitions. This is similar to weaving aspects in programming language execution, and might also suffer the lack of a higher level structural model. Process algebra is employed to model aspect-oriented software architecture [7].

3. OUR APPROACH

This section details the elements of the approach we are taking. We first give an overview of our existing architectural modeling language, and then we outline the new modeling capabilities we propose.

3.1. Overview of xADL

We extend our existing Architecture Description Language, xADL 2.0 [8], to support new modeling concepts that are necessary for secure software. xADL is an XML-based extensible ADL. It supports basic architectural constructs such as components, connectors, interfaces, and ports. It also provides an infrastructure to introduce new modeling concepts, and has been extended successfully to model software configuration management, and supports a mapping facility that links components or connectors to their implementing subarchitecture. Examples of xADL can be found in Section 4.2.2.

3.2. Modeling Architectural Security

3.2.1. Access Control: Principal, Privilege, Context

Our approach supports multiple security models that are being widely used in practice. Our first efforts are directed at the classic access control models [17], which is the dominant security enforcement mechanism. This model requires the following core concepts: *Principal, Privilege, and Context.* We extend the base xADL language with these concepts to get a new language, Secure xADL, which is still under development.

A *principal* is the user on whose behalf software executes. Principal is a key concept in security, but it is missing from traditional software architectures. A software architecture generally assumes that a) all of its components and connectors execute under the same principal, b) this principal can be determined at design time, c) it will not change during runtime, either advertently or intentionally, and d) even if there is a change, it has no impact on the software architecture. As a result, there is no modeling facility to capture allowed principals of architectural components and connectors. Also, the allowed principals cannot be checked against actual principals at execution time to enforce security conformance. We extend the basic component and connector constructs with the principal for which they perform, thus enabling architectural design and analysis based on different security principals defined by software architects.

Another important security feature that is missing from traditional ADLs is *privilege*, which describe what components can do depending on the executing principals. Current modeling approaches take a maximum privilege route, where a component's interfaces list all privileges that a component possibly needs. This is a source for privilege escalation vulnerabilities, where a less privileged component is given more privileges than what it should be properly granted. A more disciplined modeling of privileges is needed. We model two types of privileges. The first type handles the traditional resource access control, such as which principal has read/write access to which files. The second type includes architecturally important privileges, such as execution, creation and removal of new architectural components, reading and writing of architecturally critical information.

The last core concept is the *context*. When components and connectors are making security decisions, the decisions might be based on entities other than the decision maker itself. More specifically, the context of the decision making, such as the overall environment or neighboring entities, can play an important role in such decision makings. The context should also be modeled, so the security implication becomes more explicit, and any architectural changes that impact security are more apparent.

3.2.2. Components: supply security contract

The above modeling constructs (principal, privilege, and context) are currently modeled as extensions to the base xADL component types. The base xADL component types supply interface signatures, which describe the basic functionality of these components. The extended modeling constructs facilitate design and analysis of security implications for these functionalities.

3.2.3. Connectors: regulate and enforce contract

Connectors play a key role in our approach. They regulate and enforce the security contract specified by components.

Connectors can decide what principals the connected components are executing for. For example, in a normal SSL connector, the server authenticates itself to the client, thus the client knows the executing principal of the server. A stronger SSL connector can also require client authentication, thus both the server component and the client component know the executing principals of each other.

Connectors also regulate whether components have sufficient privileges to communicate through the connectors. For example, a connector can use the privileges information of connected components to decide whether a component executing under a certain principal can deliver a request to the serving component.

Connectors also have potentials to provide secure interaction between insecure components. Since many components in component-based software engineering can only be used "as is" and many of them do not have corresponding security descriptions, a connector is a suitable place to assure appropriate security. A connector decides what communications are secure and thus allowed, what communications are dangerous and thus rejected, and what communications are potentially insecure thus require close monitoring.

Using connectors to regulate and enforce a security contract and leveraging advanced connector capabilities will

facilitate supporting multiple security models [28]. These advanced connector capabilities include the reflective architectural derivation of connectors from component specifications, composing connectors from existing connectors, and replacing one connector with another connector.

4. A CASE STUDY: PROJECT IMPROMPTU

In this section we use a case study, Project Impromptu, to illustrate our initial results of applying the secure connector approach. In Section 4.1, we give an overview of the project, specifying the general context in which we make design decisions about security. Section 0 enumerates software components of the system and describes how secure connectors connect these components to accomplish security goals. Lastly, Section 4.3 illustrates how the secure connector can be replaced by another composite secure connector that achieves more security.

4.1. Overview of Project Impromptu

Project Impromptu is a subproject of Project Swirl [11]. The hypotheses of the Swirl Project are as follows. First, traditional security mechanisms must be utilized in a user-centered context to provide effective security for users. Second, users make security related decisions within a context. Different contexts require different degrees of security. Third, users' perceptions of the context can be facilitated by visualizing security related events that come from heterogeneous sources. Finally, perceptions and decisions related to security should be well integrated with users' main tasks.

Figure 1.Impromptu User Interface

Project Impromptu develops an ad-hoc file sharing application as a test bed to investigate and evaluate these hypotheses. Each Impromptu user can share files and decide how the shared files can be accessed by other users. A file can be "see-only", which means other users can only know its existence but cannot access its content. A file can be "read-only", where other users can read its content but cannot modify it. A file can also be "read-write", allowing other uses to read and modify its content. Finally, a file can be "persistent", which will still exist for read/write access even after the original owner has left the ad-hoc sharing group.

Figure 1 depicts what a user will see when Impromptu launches. The "pie" designates the entire ad-hoc file sharing group. Each slice of the pie represents a participant. The participant representing the current executing user is highlighted by the darker shaded slice. Each dot is a shared file. The position of the file determines the sharing level for each file. From the outermost ring inward, each ring represents "see-only", "read-only", and "read-write", respectively. The center circle collects all "persistent" files.

4.2. The secure architecture

4.2.1. Components and Connectors

Internally, the Impromptu application consists of the following components: the graphical user interface, the Jetty web server, the Impromptu WebDAV proxy, and the Slide WebDAV repository. The secure WebDAV connector and the YANCEES [13] event notification connector connect these components together. The architecture is depicted in Figure 2. Jetty and Slide are external open source software components. The user interface component, the proxy component, the secure WebDAV connector, and the YANCEES connector are developed by us.

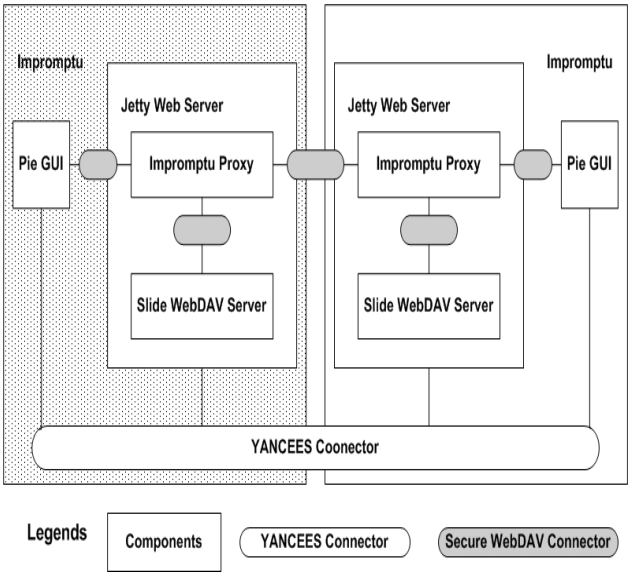

Figure 2., Impromptu Architecture

The YANCEES connector provides a high-level event notification channel. This connector delivers relevant events to interested subscribers. These events include functionality related

events, such as an indication that a file is created, and security related events, such as that the file's sharing level has been changed from "read-only" to "read-write".

Jetty serves as a dynamic application server that allows an add-on component to decide what a response will be when Jetty receives a request. Slide is an add-on component that provides WebDAV [6] repository support. WebDAV is an HTTP extension that provides Internet-scale resource storage, retrieval, and modification capability. It is an open standard, easily available in different platforms, and is thus chosen as the foundation storage for the ad-hoc file sharing application.

Participants store their own files in their own Slide server. However, this local storage is not directly seen by the participant. A user only interacts with the Impromptu proxy server, using the Pie GUI depicted in Figure 1. The proxy provides an illusion of a unified, shared file storage work space. When an Impromptu proxy receives a file operation request, it determines whether the request is directed at a local file or a remote file belonging to another participant. In the former case, it retrieves the file from the local Slide server using a standard WebDAV request. In the latter case, it performs the operation against the remote Impromptu proxy, which will accomplish the operation using its own local Slide server.

As can be clearly seen from Figure 2, the secure WebDAV connector is the key communication mechanism that connects the Slide server, the Impromptu Proxies, and the GUI. The next section outlines the security goals of the file sharing application and how the secure WebDAV connectors achieve these goals.

4.2.2. Security Goals and Mechanisms

We designed this application for a relatively friendly, ad-hoc file sharing environment. The participants are assumed to be not malicious, and the major risk in such an environment is unintentional disclosure of information. In traditional file sharing applications, when a user operates on files it is not always clear to the user what files are shared, how they might be accessed and changed, and who is currently reading and changing files. However, neither do we want to require a user to use a rather complex configuration operation to express such intentions. Such complexity might be overwhelming to the user, and thus affect usability. In summary, the security goals for the Impromptu file sharing application are 1) make security visible; 2) ease security configuration.

A type of secure WebDAV connector is designed by us to achieve the above goals. The connector employs an IP-based authentication scheme and a method-based authorization mechanism. The connectors connect the local Impromptu proxy and the Slide server, which store files that should be secured, and the GUI and the remote Proxy, which access secured files. The security architecture of a single Impromptu system is described in Figure 3, using secure xADL; some syntax details are omitted for clarity. The secure WebDAV connector type extends a base xADL connector type, ConnectorType, using the extensible feature of the xADL language. Three instances of secure WebDAV connectors connect related components.

```
<connectorType
  type="ConnectorType"
  id="SecureWebDAVConnector">
    <signature id="WebDAVClient">
    </signature>
    <signature id="WebDAVServer">
    </signature>
    <description>
        IP-based authentication
        Method-based authorization
    </description>
</connectorType>
<component type="ProxyType" id="Local">
    <principal>Me</principal>
</component>
<component type="ProxyType" id="Remote">
    <principal>Other</principal>
</component>
<component type="GUIType" id="GUI">
    <principal>Me</principal>
</component>
<component type="SlideType" id="Slide">
    <principal>Me</principal>
</component>
<connector type="SecureWebDAVConnector"
    id="GUI_Impromptu">
    <interface signature="WebDAVClient"
      id="GUI"/>
    <interface signature="WebDAVServer"
      id="Impromptu"/>
</connector>
<connector type="SecureWebDAVConnector"
  id="Impromptu_Impromptu">
    <interface signature="WebDAVClient"
      id="Remote"/>
    <interface signature="WebDAVServer"
      id="Local"/>
</connector>
<connector type="SecureWebDAVConnector"
  id="Impromptu_Slide">
    <interface signature="WebDAVClient"
      id="Local"/>
    <interface signature="WebDAVServer"
      id="Slide"/>
</connector>
```

Figure 3, Secure WebDAV Connector

The connector connects a WebDAV client and a WebDAV server. It employs two security facilities. First, the connector uses an IP-address based authentication mechanism to separate a local client from a remote client. When the connector receives a WebDAV operation request from the client, it determines, using the IP address of the client, whether the request comes from the same machine as the server (thus from the local participant), or from a different machine (thus from a remote participant). In the former case, the client component will execute as the local principal, "me". In the latter case, the client component executes as the remote principal, "other". For example, in Figure 3, connector `GUI_Impromptu` connects the GUI and the local Impromptu. The GUI executes as the "me" principal because it executes on the same machine as the local Impromptu. The connector `Impromptu_Impromptu` connects two Impromptu proxies. The remote Impromptu proxy executes under the "other" principal because it resides on a different machine than that of the local Impromptu proxy.

Second, the connector uses both the principal and the file sharing level to decide what WebDAV methods a client can perform against that file. The local GUI component, executing

as the “me” principal, can do anything towards local files. A remote participant, executing as the “other” principal, is subject to the sharing level of a file. This decision process is transparent to a user, so there is no need for the user to finish some complex setups. If a file is shared as “see-only”, the connector will only allow the WebDAV PROPFIND method to pass from the client to the server. This method permits other participants to retrieve information such as the creation date, the resource type, etc. For a “read-only” file the connector permits, in addition to the PROPFIND method, the WebDAV GET method, enabling a remote participant to get the content of a file. Finally, the secure connector permits the WebDAV PUT method for a “read/write” file, so a remote user can store back modifications for a retrieved file.

We have conducted an initial user study to assess whether the proposed security architecture can achieve the security goals [11]. The preliminary results of this study suggest that the system gives users a clearer sense of perception and manipulation of security, and it does not overwhelm users with technical details.

4.3. Planning for a revised secure architecture

The initial results of Project Impromptu are encouraging, and we are planning to widen our investigation. We are revising the security architecture to address the following concerns.

First, we are planning to deploy the Impromptu system into handheld devices, which might require a more secure authentication connector. The current Impromptu software is a tightly integrated suite of components, some of which might require too many resources to execute on handheld devices. A possible solution is to only execute the GUI on the handheld device (so we can still investigate how users perform their regular and security tasks on such hardware platforms), and deploy the rest of the Impromptu software on more powerful platforms. Under such a configuration, the IP-address based authentication mechanism will be insufficient, because even requests from the owning participant (who is using a handheld device) will come from a different IP-address. We are planning to adopt a more secure authentication connector, such as a digest authenticator. Such a connector enables deploying the Impromptu system to an environment that might contain malicious adversaries, and mitigate some limitations of an address-based authentication mechanism [2].

Second, we are planning to utilize existing authorization mechanisms included in Jetty and Slide to enable richer semantics. Two extensions to the current method-based authorization connector are possible: better integration with mechanisms provided by the Jetty application server and the Slide WebDAV server, and more leverage of the standard WebDAV ACL [6] access control features provided by Slide.

The aforementioned secure WebDAV connector in Section 4.2.2 could be replaced with the following connector, which consists of a digest authentication connector, a standard authorization connector using web.xml deployment descriptor [5], and a standard WebDAV ACL authorization connector. This illustrates composing a secure connector from several sequential secure sub-connectors.

```
<connectorType
  id="DigestAuthenticationConnector">
</connectorType>
<connectorType
  id="WebXMLAuthorizationConnector">
</connectorType>
<connectorType
  id="WebDAVACLConnector">
</connectorType>
<connectorType
  id="SecureWebDAVConnector">
   <subArchitecture>
     <sequence>
     <connector type=
       "DigestAuthenticationConnector"/>
     <connector type=
       "WebXMLAuthorizationConnector"/>
     <connector type=
       "WebDAVACLConnector"/>
     </sequence>
   </subArchitecture>
</connectorType>
```

Figure 4, Planned Secure WebDAV Connector

5. CONCLUSION

Component-based software operating in a modern networked environment presents new challenges that have not been fully addressed by traditional security research. Recent research on software architecture shed light on high-level structure and communication issues, but has paid insufficient attention to security.

We argue that a connector-based secure software architecture technology is necessary to advance existing knowledge and meet the new challenges. We extend component specifications with core security concepts: principal, privilege, and context. Their compositions are handled by connectors, which regulate security policy. We illustrate our approach through a component-based secure file sharing application.

This research is still on-going work. The contributions of this research lie in that 1) we address the security problem from an architectural viewpoint. Previous experiences have shown that traditional security infrastructure is insufficient in providing comprehensive security. Our use of an architecture model can guide the design and analysis of secure software systems based on such infrastructure; 2) we provide an approach for describing and constructing secure connectors, which are the central vehicle for secure interaction of complex software components.

Our future work includes 1) extending support for the classic Access Control model; 2) supporting the newer Role-based access control model [25] and the trust management model [30]; 3) developing a set of tools (visual modeling, implementation generation, and light-weight formal analysis) to support modeling architectural security. These tools will extend our existing development environment, ArchStudio [8].

6. ACKNOWLEDGEMENTS

The Impromptu project is developed by the Swirl group. The authors would like to thank Ben Pillet, Xianghua Ding, Kari

Nies, Roberto Silva Filho, Rogerio DePaula, Jennifer Rode, and John Georgas for their wonderful work and insightful comments.

This work was supported in part by the National Science Foundation under award 0326105, and by a grant from Intel Corporation.

7. REFERENCES

[1] Allen, R. and Garlan, D., *A Formal Basis for Architectural Connection.* ACM Trans. Softw. Eng. Methodol., 1997. **6**(3): p. 213-249.

[2] Bellovin, S.M., *Security Problems in the Tcp/Ip Protocol Suite.* ACM SIGCOMM Computer Communication Review, 1989. **19**(2): p. 32-48.

[3] Berghel, H., *The Code Red Worm.* Communications of the ACM, 2001. **44**(12): p. 15-19.

[4] Bidan, C. and Issarny, V. *Security Benefits from Software Architecture*. in Proceedings of 2nd International Conference on Coordination Languages and Models, p.64-80, 1997.

[5] Bodoff, S., Armstrong, E., Ball, J., Carson, D., Evans, I., and Green, D., *The J2ee™ Tutorial*. 2nd Edition ed. 2004: Addison-Wesley Professional.

[6] Clemm, G., Reschke, J., Sedlar, E., and Whitehead, J., *Web Distributed Authoring and Versioning (Webdav) Access Control Protocol.* RFC 3744, 2004.

[7] Cuesta, C.E., Romay, M.P., Fuente, P.d.l., and Barrio-Solorzano, M. *Reflection-Based, Aspect-Oriented Software Architecture*. in Proceedings of 1st European Workshop on Software Architecture, p.43-56, 2004.

[8] Dashofy, E.M., van der Hoek, A., and Taylor, R.N. *An Infrastructure for the Rapid Development of Xml-Based Architecture Description Languages*. in Proceedings of Proceedings of the 24th International Conference on Software Engineering, p.266-276, 2002.

[9] DeLine, R., *Avoiding Packaging Mismatch with Flexible Packaging.* IEEE Transactions on Software Engineering, 2001. **27**(2): p. 124-143.

[10] Deng, Y., Wang, J., Tsai, J.J.P., and Beznosov, K., *An Approach for Modeling and Analysis of Security System Architectures.* IEEE Transactions on Knowledge and Data Engineering, 2003. **15**(5): p. 1099-1119.

[11] DePaula, R., Ding, X., Dourish, P., Nies, K., Pillet, B., Redmiles, D., Ren, J., Rode, J., and Filho, R.S., *In the Eye of the Beholder: A Visualization-Based Approach to Information System Security.* Submitted to International Journal of Human-Computer Studies, 2005.

[12] Ducasse, S. and Richner, T. *Executable Connectors: Towards Reusable Design Elements*. in Proceedings of 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering, p.483-499, 1997.

[13] Filho, R.S.S., Souza, C.R.B.d., and Redmiles, D.F. *The Design of a Configurable, Extensible and Dynamic Notification Service*. in Proceedings of 2nd International Workshop on Distributed Event-based Systems, p.1-8, 2003.

[14] France, R., Ray, I., Georg, G., and Ghosh, S., *Aspect-Oriented Approach to Early Design Modelling.* IEE Proceedings-Software, 2004. **151**(4): p. 173-185.

[15] Jürjens, J. *Umlsec: Extending Uml for Secure Systems Development*. in Proceedings of UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language, p.412--425, 2002.

[16] Katara, M. and Katz, S. *Architectural Views of Aspects*. in Proceedings of Proceedings of the 2nd international conference on Aspect-oriented software development, p.1-10, 2003.

[17] Lampson, B.W., *A Note on the Confinement Problem.* Communications of the ACM, 1973. **16**(10): p. 613-15.

[18] Lodderstedt, T., Basin, D.A., J, and Doser, r. *Secureuml: A Uml-Based Modeling Language for Model-Driven Security*. in Proceedings of UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language, p.426--441, 2002.

[19] Lopes, A., Wermelinger, M., and Fiadeiro, J.L., *Higher-Order Architectural Connectors.* ACM Transactions on Software Engineering and Methodology, 2003. **12**(1): p. 64-104.

[20] Magee, J. and Kramer, J. *Dynamic Structure in Software Architectures*. in Proceedings of Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering, p.3-14, 1996.

[21] Medvidovic, N. and Taylor, R.N., *A Classification and Comparison Framework for Software Architecture Description Languages.* Software Engineering, IEEE Transactions on, 2000. **26**(1): p. 70-93.

[22] Mehta, N.R., Medvidovic, N., and Phadke, S. *Towards a Taxonomy of Software Connectors.* in Proceedings of 22nd International Conference on Software Engineering, p.178-187, 2000.

[23] Moriconi, M., Qian, X., Riemenschneider, R.A., and Gong, L. *Secure Software Architectures.* in Proceedings of 1997 IEEE Symposium on Security and Privacy, p.84-93, 1997.

[24] Ray, I., France, R., Li, N., and Georg, G., *An Aspect-Based Approach to Modeling Access Control Concerns.* Information and Software Technology, 2004. **46**(9): p. 575-587.

[25] Sandhu, R.S., Coyne, E.J., Feinstein, H.L., and Youman, C.E., *Role-Based Access Control Models.* Computer, 1996. **29**(2): p. 38-47.

[26] Spitznagel, B. and Garlan, D. *A Compositional Approach for Constructing Connectors*. in Proceedings of 2nd Working IEEE/IFIP Conference on Software Architecture, p.148-157, 2001.

[27] Sun, W. and Dai, Z. *Aosam: A Formal Framework for Aspect-Oriented Software Architecture Specifications.* in Proceedings of The 8th IASTED International Conference on Software Engineering and Applications, 2004.

[28] Tisato, F., Savigni, A., Cazzola, W., and Sosio, A. *Architectural Reflection. Realising Software Architectures Via Reflective Activities*. in Proceedings of 2nd International Workshop on Engineering Distributed Objects, p.102-15, 2000.

[29] Wing, J.M., *A Call to Action: Look Beyond the Horizon.* Security & Privacy Magazine, IEEE, 2003. **1**(6): p. 62-67.

[30] Winslett, M. *An Introduction to Trust Negotiation*. in Proceedings of 1st International Conference on Trust Management, p.275-283, 2003.