Developing Secure Architectures for Middleware Systems

Eduardo B. Fernandez and Maria M. Larrondo-Petrie
Florida Atlantic University, Department of Computer Science and Engineering
Boca Raton, USA, 33431-0991
ed@cse.fau.edu, petrie@fau.edu

Abstract

We increasingly rely on complex applications that are typically distributed and implemented in systems that must have high reliability and security. Some of these applications, e.g., medical, financial, military, and legal, additionally require compliance with regulatory standards. Integration of these applications is achieved using a Web Application Server, a type of middleware with a global enterprise model. We consider the security needed to support such type of middleware, present patterns that can be used to build secure middleware, and show how to combine them to provide security to specific functions. We see the secure architecture as a composition of functional (unsecured) patterns with patterns that provide specific security functions. We show in some detail how we can start from general distribution and component patterns and add security patterns to build a secure middleware architecture.

Keywords: Security, Software Architecture, Design Patterns, Middleware.

1. INTRODUCTION

A variety of complex applications have become very important recently. These include medical, financial, military, and legal applications, among others. These applications are typically distributed and implemented in systems that have additional non-functional requirements such as reliability or fault tolerance. Their implementation requires a variety of units, some built ad hoc and some bought or outsourced. Another typical aspect of these systems is that they must follow regulatory standards, e.g. HIPAA [1], Sarbanes/Oxley [2], or military standards. These systems may include several databases of different types and most likely require Internet access. The applications in these systems are usually integrated using a Web Application Server (WAS), a type of middleware that has a global enterprise model, typically implemented using object-oriented components such as J2EE or .NET. These applications are of fundamental value to enterprises and their security is extremely important. We look here at some security aspects of the middleware structure needed to support such an environment.

Embedding security into middleware systems requires a secure development methodology. We have proposed a methodology that helps developers build secure systems without being security experts [3]. This methodology accomplishes its purpose through the use of patterns. Our discussion here is independent of this methodology although this work resulted as a consequence of that work. We concentrate on system architecture aspects; network security relies heavily on cryptography and is not considered in our discussion. Agents are also of interest in this context but are not considered either.

Patterns provide solutions to recurrent problems and many of them have been catalogued. We see the use of patterns as a fundamental way to incorporate security principles in the design process even by people having little experience with security practices. In our work we have produced many security patterns, e.g., [4, 5, 6]. We also developed a type of pattern called Semantic Analysis Pattern (SAP), which implements a set of basic use cases [7]. We have shown that we can combine SAPs and security patterns in a natural way to create authorized SAPs, which can be converted into models for secure designs. SAPs can be used to build conceptual models in which the necessary security constraints can be defined. We have also addressed how to carry over the security model of the analysis stage into the design stage [8]. We show here how patterns allow us to define a secure architecture for middleware systems, able to accommodate very strict requirements. A complete overview of our work so far can be found in [3].

We see the secure architecture as a composition of functional (unsecured) patterns with patterns that provide specific security functions. We show in some detail how we can start from general distribution and component patterns and add security patterns to build secure middleware architectures.

There has been a good amount of work on software architectures for middleware systems [9]. However, there much less work about their security. Schmidt studies the use of patterns to build extensible brokers [10] and to build telecommunications systems [11], but he does not consider security aspects, although his more recent papers consider security [12]. Crane et al. [13] consider patterns for distribution but again they do not include security aspects. Keller et al. [14] discuss patterns for network management but they don't include security. Security aspects are considered in [15] that analyzes how to combine security policies in heterogeneous middleware, and [16] that defines how to find identities for clients and servers. [17] applies Aspect-oriented programming to separate middleware services, including security. These papers consider very specific security problems, we are interested in the global security architecture. Global security aspects are discussed in [18], which focuses in the combination of RBAC and multilevel models but does not consider architectures using patterns.

Section 2 presents an overview of our approach. Section 3 discusses security aspects of components. Section 4 considers security in the distribution architecture, while the last section presents some conclusions.

2. MIDDLEWARE AND SECURITY

Security patterns provide an interesting way to represent the structure of a secure system. Specifically, a security pattern describes a solution to a recurrent security problem in a given context [19]. The functional aspects of a middleware system have been described by patterns [20, 21, 9]. We can extend those patterns to incorporate security aspects as shown in Figure 1. The patterns are defined in the specific architectural layers where the information needed for security decisions is available. A basic principle of security states that all architectural levels must be covered to have a secure system [22]. To this principle we add the principle that every pattern must incorporate security features to have a secure system. The isolated patterns Secure Layers, Secure Façade, and Secure Reflection are orthogonal and can be applied to any of the other patterns. Starting from the conceptual model of the application (maybe composed from a set of patterns) we define security constraints (rules) at that level. These rules are stored in a Policy Administration Point (PAP) [23]. These rules are enforced when a request is sent to the Policy Enforcement Point (PEP) that consults the Policy Decision Point (PDP). The PDP uses the information in the PAP and in the Policy Information Point (PIP) to decide if the request is valid. A concrete implementation of policies is described in [24].

The next architectural level includes architectural patterns describing the software architecture of the application. The standard functional patterns for this level have been complemented with security functions and we have then: Secure Model View Controller, Secure Adapter, Secure Broker, Security enterprise Component Framework, and Secure Web Services.

The following level corresponds to databases and application-level communications. Here we have a secure Relational Database Mapping or Mapper, a Secure Proxy, and a Secure Client/Dispatcher/Server.

The final level includes a Secure Operating System, a Secure Channel, and an Authenticator.

We discuss in the next section how to add security to functional patterns to obtain their secure versions. To illustrate the approach we consider two aspects of basic importance for the security of a middleware architecture:

- How to store and execute a business enterprise model. Business models are handled through component frameworks, typically using an object-oriented model. This model may consume or provide web services.

- Its distributed systems architecture. Distribution is handled through distributed objects or web services protocols.

Clearly, there are other security aspects that are also important but those will be considered in later work.

3. COMPONENTS AND SECURITY

Several patterns solve specific problems of components:

- The *Enterprise Component Framework* pattern [25], describes the container structure of components. This representation of components can describe J2EE and .NET components by proper specialization.

- The *Component Configurator* [9] lets an application dynamically attach and detach components or processes.

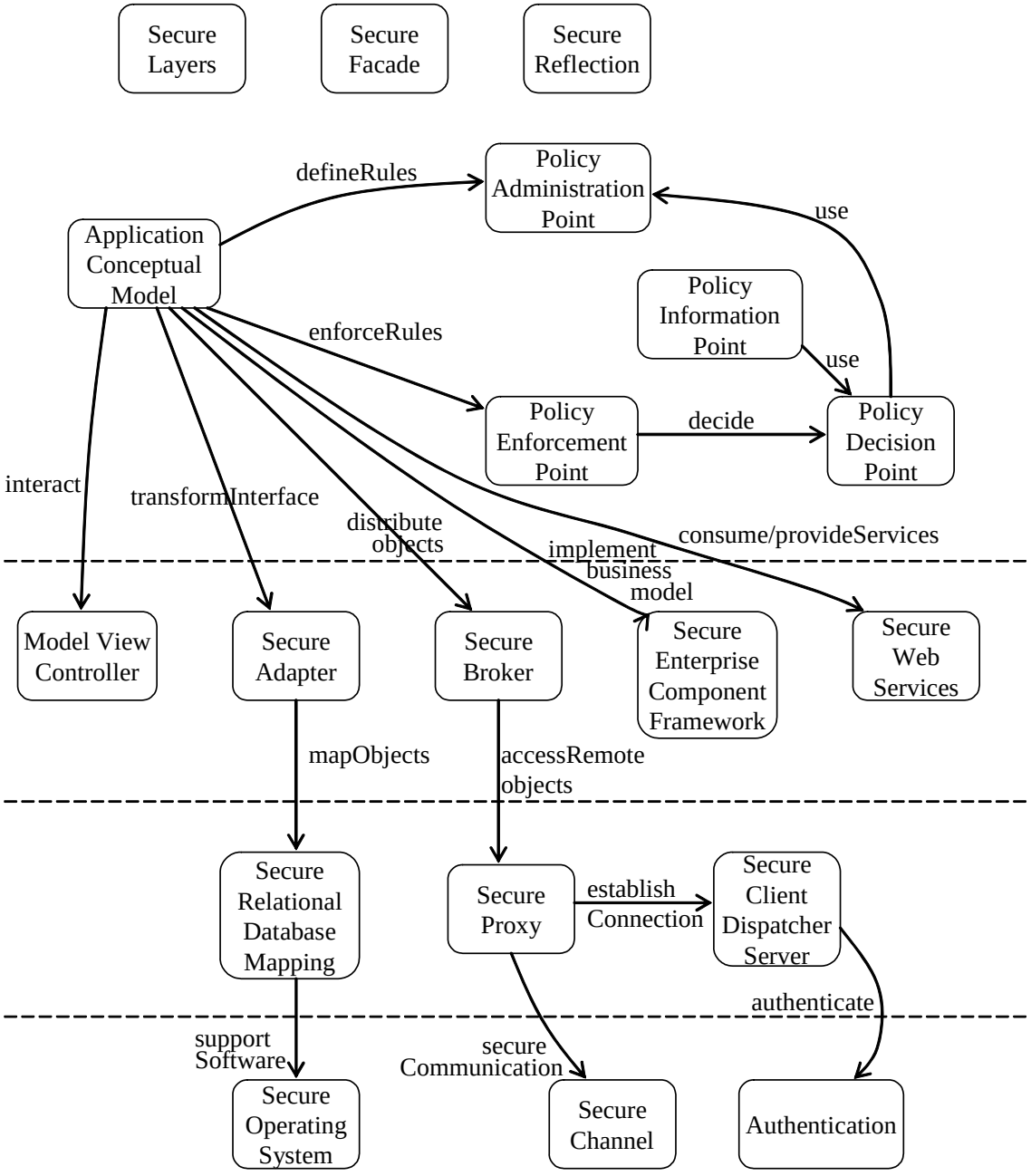

Figure 1. Pattern diagram for middleware architectural patterns.

- The *Interceptor* [9] allows the transparent addition of services to an application or framework. These services are automatically invoked when certain events occur.
- The *Extension Interface* [9] defines multiple interfaces for a component.
- The *Home* pattern separates the management of components from their use by defining an interface for creating instances of components.

Three patterns are used to handle persistent data and to hide low-level details:

- The *Façade* [21] provides a unified, higher-level interface to a set of interfaces in a subsystem.
- The *Adapter* [21] converts the interface of an existing class into a more convenient interface.
- The *Wrapper Facade* [9] encapsulates the functions and data provided by existing subsystems or levels and defines a higher-level interface.

First we add security to the component patterns:

- The *Enterprise Component Framework* can include security descriptors that define authorization rules (Figure 2). These rules can then be enforced by a concrete version of the Reference Monitor pattern [6].
-

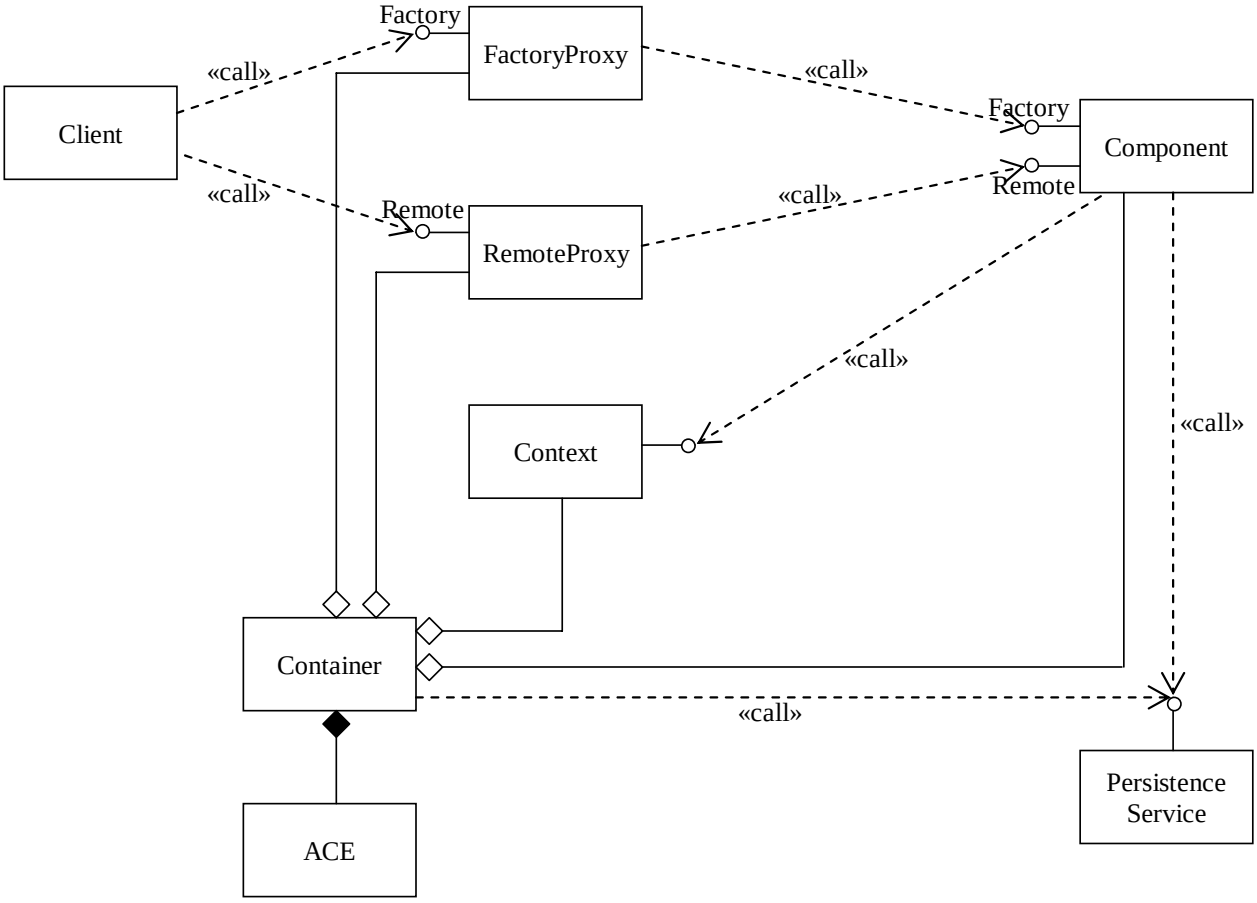

Figure 2. Class diagram of the Enterprise Component Framework pattern.

- The *Component Configurator* can be used to reduce the time when critical processes are exposed to attacks by hiding them from the visibility of suspicious processes. There could be versions of some modules with different levels of security; more secure versions could be configured after some events, e.g. a security alert from an Intrusion Detection System.
- The *Interceptor* is useful to add security functions to a framework, e.g. a CORBA-based system, if the original implementation did not have them. The intercepted requests can be checked by a concrete version of the Reference Monitor; e.g. the one described in [24].

- The *Extension Interface* can be used to define views that let a user or role access only some parts of the information in specific ways, according to their authorizations. This is similar to the use of views to control access to relational databases [22].

- The *Home* pattern can be used to apply authorization rules to control the creation of objects in components. When a new object is created, we should add to it the list of the subjects that have some access to it, including their type of access. Unrestricted numbers of created objects could allow an attacker to perform denial of service. This control has been done in some operating systems [6].

Similarly, we can add security patterns to the three other patterns (Figure 3):

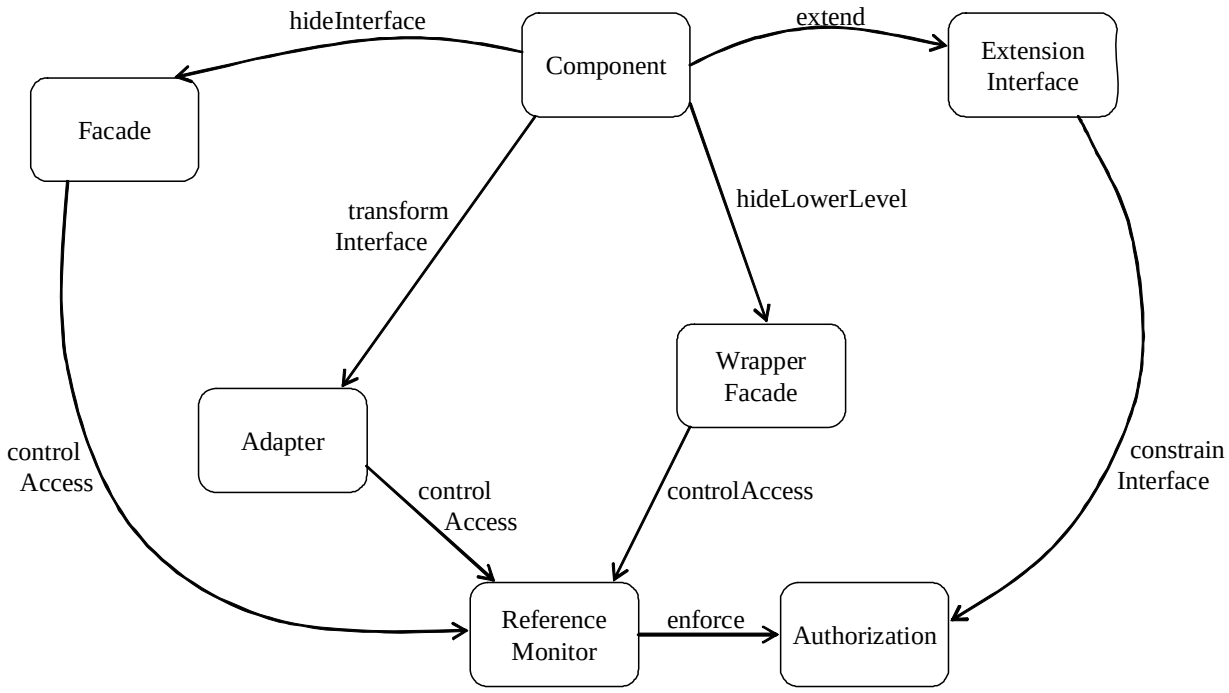

Figure 3. Pattern diagram for securing hiding patterns.

- The *Façade* can hide implementation details that could be exploited by hackers and can apply access control of the operations of the Façade according to authorization rules.

- The *Adapter* can be used to define a new interface with fewer operations for some uses according to their security restrictions, to map database security constraints to application constraints (or vice versa), or to just control access to the operations of the interface.

- The *Wrapper Façade* can be used to hide the implementation of the lower levels. This prevents attackers form taking advantage of implementation flaws. A higher-level interface restricts the possibilities of a hacker. Access control is also possible to the operations of the interface.

Figure 4 summarizes these extensions showing how a Component pattern can be composed with other patterns that provide security services. Figure 3 shows a similar summary for the hiding patterns.

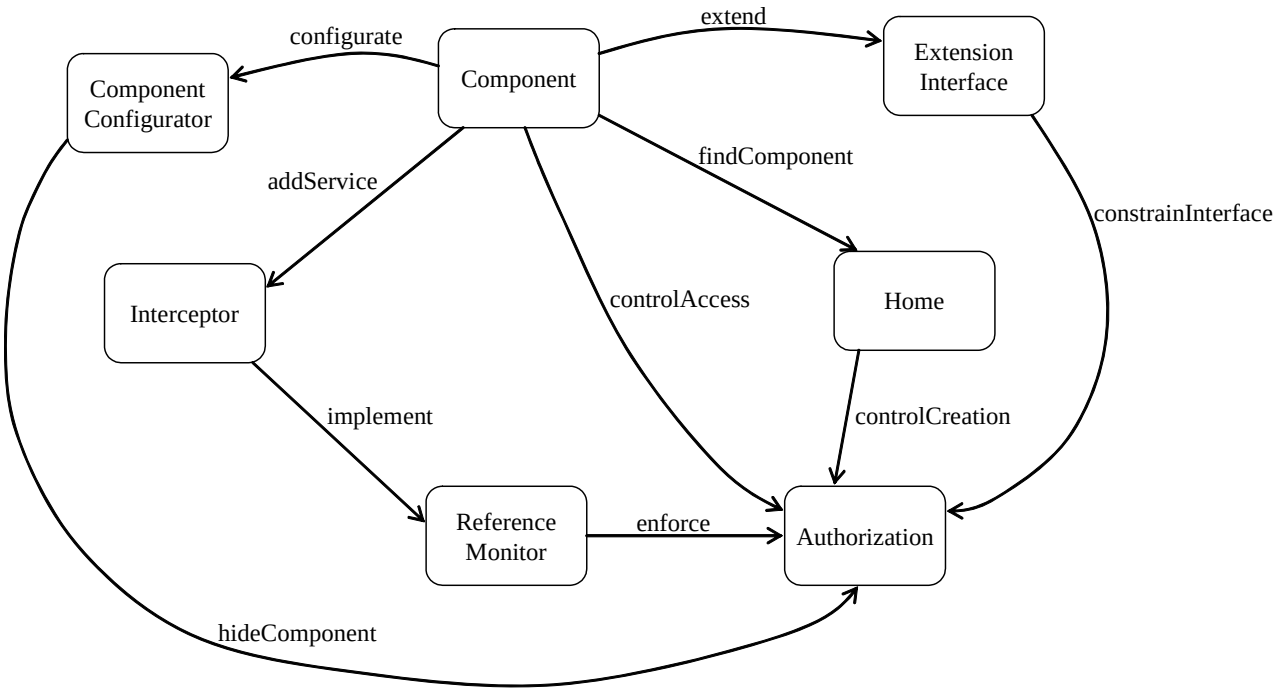

Figure 4. Pattern diagram for securing component patterns.

4. DISTRIBUTION ARCHITECTURE AND SECURITY

Figure 5 shows how we can add security to the Broker pattern by composing it with several security patterns. A Secure Broker provides transparent and secure interactions between distributed components.

Interpreting the meaning of the patterns we have to apply access control at the local proxy and the object adapter and we also need to define a secure channel between the client and the server. We need also mutual authentication between the local proxy and the object adapter. We show two possibilities for authorization (Access matrix and RBAC); we did not do this in the earlier figures for simplicity.

The Client Dispatcher Server pattern is, in turn, implemented using Authenticator, Lookahead, Connector/acceptor, and other lower-level patterns. These may apply some of the required enforcement; for example, the Connector when establishing a new connection would apply authentication.

There are also many standards for web services security, e.g., XACML [23] and SAML [26]. They must be considered when producing or consuming web services in the middleware. Their combination with the remote object security architecture makes middleware security quite complex. For example, a pattern for XACML [5] can be combined with functional web service patterns.

An important direction is wireless devices using web services. A cell phone can be a consumer and provider of web services. Web services standards can be transported to wireless devices by appropriately reducing them [27]. The architectures used for remote objects do not appear very appealing for wireless devices and we know of no commercial system using this distribution approach. However, it may be of interest for systems that require a higher level of security or performance or that will be used in restricted environments, e.g. within a company.

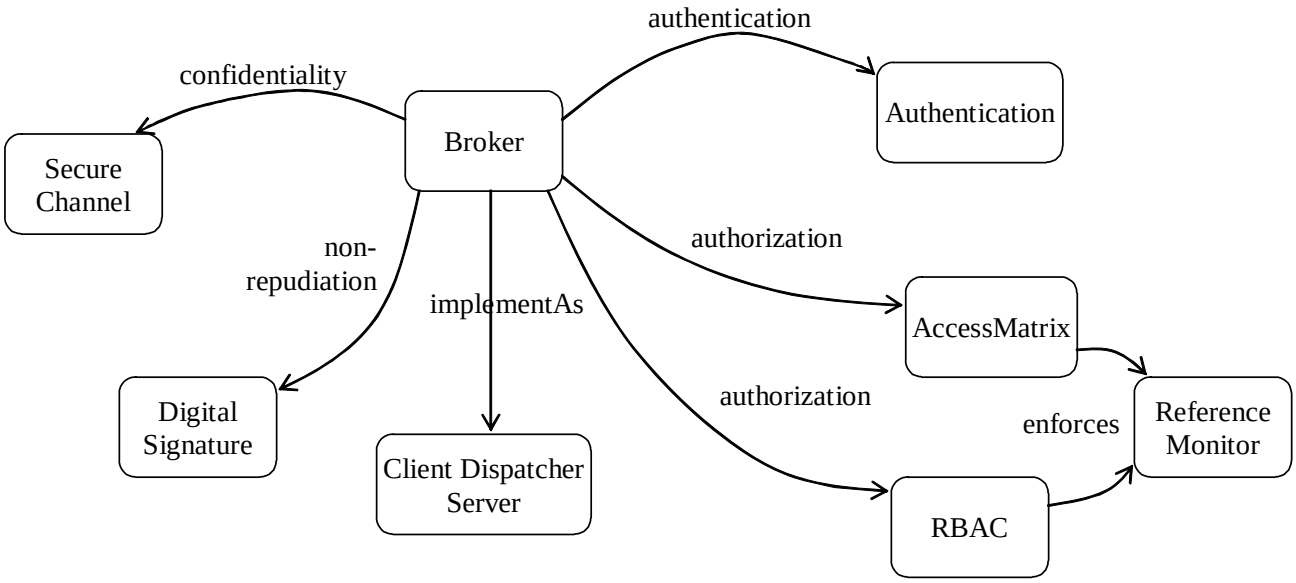

Figure 5. Pattern diagram for Secure Broker.

5. CONCLUSIONS

We have analyzed how the security of a typical middleware system, a Web Application Server, can be enhanced by securing its subsystems. In particular, we considered a system with distributed objects. Systems using web services can be analyzed similarly except that they use a larger variety of security standards, which implies a larger variety of patterns. Web services have a strong affinity to remote objects [28]. They need life cycle support, dynamic object creation/deletion, state management, transaction support. While most current web services platforms don't provide this, it is clear that these functions will be in them in the future. This implies an evolution of middleware systems where combinations of patterns like these or new patterns will be needed. The use of patterns can provide the required level of flexibility.

Some of these patterns have been written already (by us or others) while others are future work. Until now there are patterns for Secure Layers [6], Secure Operating Systems [6], Secure Channel [29], Authentication [6], and Secure Broker [30].

We indicated that these patterns should be used through a specific methodology and we have proposed an approach of this type [3]. That approach and the patterns presented here could be the basis of a specific approach to build secure middleware.

Acknowledgements

The referees made useful comments that improved this paper. This work was supported by a grant from the US Dept. of Defense (DISA), administered by Pragmatics, Inc.

References

[1] Health Insurance Portability and Accountability Act of 1996. http://www.hipaa.org/

[2] One Hundred Seventh Congress of the United States of America, "Sarbanes-Oxley Act of 2002", January 23, 2002, http://news.findlaw.com/hdocs/docs/gwbush/sarbanesoxley072302.pdf

[3] Fernandez, E.B. Larrondo-Petrie, M.M., Sorgente, T., and VanHilst. M. A methodology to develop secure systems using patterns", to appear as Chapter 5 in the book *Integrating security and software engineering: Advances and future vision*, H. Mouratidis and P. Giorgini (Eds.), IDEA Press, 2006.

[4] Fernandez, E.B. and Pan, R. A Pattern Language for security models. *Procs. of the 8th Annual Conference on Pattern Languages of Programs (PLoP 2001),* 11-15 September 2001, Allerton Park Monticello, Illinois, USA,

2001. http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions

[5] Delessy, N., and Fernandez, E.B. Patterns for the eXtensible Access Control Markup Language. *Procs. of the Pattern Languages of Programs Conference (PLoP 2005)*, 7-10 September 2005, Allerton Park Monticello, Ilinois, USA, 2005.

[6] Schumacher, M., Fernandez, E.B., Hybertson, D, Buschmann, .F. and Sommerlad, P. *Security Patterns: Integrating security and systems engineering*. West Sussex, England: John Wiley & Sons. 2006.

[7] Fernandez, E.B. and Yuan, X. Semantic analysis patterns. *Procs. of 19th Int. Conf. on Conceptual Modeling / the Entity Relationship Approach, ER2000*, 183-195. Also available from: http://www.cse.fau.edu/~ed/SAPpaper2.pdf

[8] Fernandez, E.B., Sorgente, T. and.Larrondo-Petrie, M.M. A UML-based methodology for secure systems: The design stage. *Procs. of the Third International Workshop on Security in Information Systems (WOSIS-2005)*, Miami, Florida, USA, 24-25 May 2005, pp. 207-216.

[9] Schmidt, D. C., Stal, M., Rohnert, H., and Buschmann, F. *Pattern-Oriented Software Architecture*," West Sussex, England: John Wiley & Sons. 2000.

[10] Schmidt D.C. and Cleeland, C. Applying patterns to develop extensible ORB middleware. *IEEE Communications Magazine*. Vol. 37, No. 4, (April 1999). Los Alamitos, California, USA: IEEE Computer Society. 1999, pp. 54-63.

[11] Schmidt, D.C. Using design patterns to develop reusable object-oriented communication software. *Communications of the ACM*, Vol. 38, No 10, (October 1995). Sausalito, California, USA: ACM. 1995, pp. 65-74.

[12] Wang, N. Parameswaran, K. and Schmidt, D.C. The design and performance of meta-programming mechanisms for object request broker middleware. *Procs. of the 6th Usenix Conf. on Object Oriented technology and Systems (COOTS).* 29 January – 2 February 2001. San Antonio, Texas, USA. http://www.usenix.org/events/coots01/wang.html

[13] Crane, S., Magee, J., and Pryce, N. Design patterns for binding in distributed systems. *OOPSLA'95 Workshop on Design Patterns for Concurrent, Parallel and Distributed Object-Oriented Systems*. October 1995. Austin, Texas, USA. 1995.

[14] Keller, R.K., Tessier, J. and von Bochmann, G. A pattern system for network management interfaces. *Communications of the ACM*, Vol. 41, No 9, September 1998, Sausalito, California, USA: ACM. 1998, pp. 86-93.

[15] Foley, S.N., Quillinan, T.B., O'Connor, M., Mulcahy, B.P. and Morrison, J.P. A framework for heterogeneous middleware security. *Procs. of the 13th heterogeneous computing workshop (HCW'04)*. 26 April 2004. Santa Fe, New Mexico, USA. Los Alamitos, California, USA: IEEE Computer Society, 2004, pp. 1-11.

[16] Lang, U., Gollmann, D. and Schreiner, R. Verifiable identifiers in middleware security. *Proceedings of the 17th Annual Computer Security Applications Conference, ACSAC*, 10-14 December 2001. New Orleans, Louisiana, USA. Los Alamitos, California, USA: IEEE Computer Society, 2001.

[17] Eichberg, M. and Mezini, M. Alice: Modularization of middleware using Aspect-Oriented programming. *Procs. of the Conf. on Software Eng. and Middleware (SEM)* 2004, Linz, Austria, Sept. 2004. Vol. 3437 of Lecture Notes in Computer Science. London, England: Springer-Verlag, 2004, pp. 47-63.

[18] Demurjian, S., Bessette, K., Doan, T. and Phillips. C. Concepts and Capabilities of Middleware Security. Chapter 9 of *Middleware for Communications*, Qusay H. Mahmoud (editor). West Sussex, England: John Wiley & Sons, 2004, pp. 211-236.

[19] Fernandez, E.B. and Larrondo-Petrie, M.M. Security patterns and secure systems design". *Procs. of the Latin American and Caribbean Conference for Engineering and Technology (LACCEI 2006)*, L.A. Godoy, M.M. Larrondo-Petrie, I. Pagan-Trinidad (Eds.) Mayagüez,, Puerto Rico, June 21-23, 2006. Mayagüez, Puerto Rico: Editorial ELIYAN, 2006.

[20] Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P. and Stal., M. *Pattern-oriented software architecture*. West Sussex, England: John Wiley & Sons. 1996.

[21] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, Massachussetts, USA: Addison-Wesley. 1994.

[22] Fernandez, E.B., Gudes, E. and Olivier, M. To appear *The design of secure systems*. Reading, Massachussetts: Addison-Wesley, 2006.

[23] OASIS eXtensible Access Control Markup Language (XACML) - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[24] Lang, U. and Schreiner, R. Integrated IT security: Air-traffic management case study, *Procs. ISSE 2005 – Securing Electronic Business Processes: Highlight of the Information Security Solutions Europe 2005 Conference*. 19-22 May 2005, Wiener Neustadt, Austria, S. Paulus, N. Pohlmann and H. Reimer, Eds. London, England: Springer Verlag. 2005.

[25] Kobryn, C. Modeling components and frameworks with UML. *Communications of the ACM*. Vol. 43, No. 10, (October 2000). Sausalito, California, USA: ACM. 2000, pp. 31-38.

[26] OASIS Security Services (SAML), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[27] Chu, H-H., You, C-W., and Teng, C-M. Challenges: Wireless Web Services. *Procs of the 10th International Conference on Parallel and Distributed Systems (ICPADS '04).* 7-9 July 2004. Los Alamitos, California, USA: IEEE Computer Society Press. 2004, pp. 657-664

[28] Birman, K.P. Viewpoint: Like it or not, web services are distributed objects. *Communications .of the ACM*. Vol. 47, No 12, (December 2004), Sausalito, California: ACM. 2004, pp. 60-62.

[29] Braga, A.M., Rubira, C.M.F. and Dahab, R. Tropyc: A pattern language for cryptographic software. Chapter 16 in *Pattern Languages of Program Design 4* (PLOPD4). 13-16 August 2000. Allerton Park Monticello, Illinios, USA. N.D. Harrison, B. Foote, and H. Rohnert, (Eds.) Reading, Massachussetts: Addison Wesley. 2000, pp. 337-371.

[30] Morrison, P. and Fernandez, E.B. The Secure Broker Pattern. *Proceedings of the 11th European Conference on Pattern Languages of Programming (EuroPLoP 2006), 5-9* July 2006. Irsee, Germany. 2006.