Adding Secure Communication Mechanism to Existing Distributed Applications by means of AOP

Ozgur Koray Sahingoz

Abstract The object-oriented programming paradigm is a process of implementing a program by means of *'Objects'* into which separate concerns are grouped. However, it does not map some types of concerns such as security, logging and exception handling, which should be implemented in each object separately. As most of the security goals, reliable communication is a non-functional requirement in a distributed system development process, and it typically crosscuts many objects in the distributed architecture. Program codes to realize this secure communication goal is generally spread in different code places throughout the application. Aspect-oriented programming (AOP) is a new programming paradigm that improves program modularity by enabling the separation of concerns from the main logic of the application. For example, in the context of security, developers should not need to encode security logic in the main program; instead, it can be grouped into a separate and independent unit, called as *aspects*. This paper presents a case study to illustrate how aspect oriented approach can be used to resolve the scattered and tangled concerns*, like secure communication,* in a previously developed distributed system in which objects communicate with each other via Java RMI. As a java-based aspect oriented tool, AspectJ is used to encapsulate the security related crosscutting concerns like communication. Performance evaluations are tested for adding security aspect to a distributed application. As a result, usage of aspects is a good choice for enhancing system to achieve high cohesion and low coupling, which are one of the main the software engineering requirements. It also enhances the readability of the system and makes system easier to maintain.

O. K. Sahingoz (✉)
Turkish Air Force Academy, Computer Engineering Department, 34149 Istanbul, Turkey
e-mail: sahingoz@hho.edu.tr

Y.-H. Han et al. (eds.), *Ubiquitous Information Technologies and Applications*,
Lecture Notes in Electrical Engineering 214, DOI: 10.1007/978-94-007-5857-5_1,
© Springer Science+Business Media Dordrecht 2013

1 Introduction

Software maintenance is a widely known problem in the software industry, and it consumes a large part of the overall software lifecycle. Some studies indicate that 50–90 % of lifecycle costs are spent on software maintenance. The object-oriented programming paradigm is one of the most used strategies of software development for organizing functional and distributed applications. However, it does not map some types of concerns such as *security, logging, and exception handling,* which are scattered into different objects in the distributed application separately.

As other concerns, *security* is a critical issue for distributed and large-scale software systems, especially which are connected to the Internet. Most of these type systems suffer from various malicious attacks, and therefore, security is a specific and important topic, and its logic should be implemented by security engineers.

Secure communication is the main solution for protecting system from this type of attacks. In distributed systems, objects generally communicate with each other by message passing. For message passing, distributed systems generally use remote procedure call mechanisms that are procedural invocation from an object on one machine to an object on another machine. Many of these systems communicate via Java RMI that is a lightweight networking technology proposed by Sun Microsystems.

Especially, most of the previously developed distributed systems do not take care of secure messaging. For adding necessary security mechanisms, developers should analyze the whole application codes *line-by-line* and should find each relevant remote procedure call methods. Same problem is also valid for Java RMI, and one limitation of this mechanism is that it not only sends simple variable types like integer, float, char, etc. but also user defined serializable objects. To encrypt these parameters, programmers should change the types of parameters. For example, an *"int"* parameter cannot be send as *"encrypted int"*, or a *"char"* parameter cannot be send as *"encrypted char"* with the *same encryption algorithm* in a modular way. Therefore, there is a need of a new approach to this problem and update security descriptions of the system in a modular way.

Security descriptions are crosscutting in a distributed system, and the developers have to write security related codes mixed with main application code [1]. However, it is also a scattering concern and most security related codes are scattered between different objects. It is a challenging issue to make a better modularized system by grouping these concerns into a unit. If it can be modularized in a well-defined way, undoubtedly, development, maintenance, and evolution of the system will be easier.

Aspect-Oriented Programming (AOP) approach is an emerging software development paradigm that is a suitable solution in software engineering paradigm and it enables improved separation of concerns by modularizing crosscutting concerns in a unit, called as aspect. By this approach, AOP not only improves the system's modularity but also decreases the system's complexity.

Many works have already been done on solving security issues with AOP. Stevenson et al. [2] tried to convert the proxy object of RMI to smart proxies by using dynamic distributed aspect-oriented programming with the Java Aspect Components system; Bostrom [3] developed a real-life healthcare application by applying database encryption with AOP methodologies; Kotrappa and Kulkarni [4] presented an aspect-oriented approach for enabling multilevel security system in which it is aimed to provide users to share information in a classification based on a system of hierarchical security levels; finally Yang et al. [5] have proposed AspectKE*, which is an AOP language based on a distributed tuple space system under Java environment, and it uses two staged implementation strategy for gathering static analysis of the system.

In the previous work of this research [6], an aspect oriented communication mechanism is constructed in a distributed system, which makes connections and communications over Java RMI with only limited parameter types. In this paper, it is aimed to extend the secure communication mechanism to transfer all type of objects. Usage of AOP allows separation of security mechanisms from the system's main functional components and enables programmers to focus on implementation of the main application. At the same time, security experts can implement the security properties separately. This approach enables system update by only developing necessary aspect definitions, and distributed system not only is modularized well but also there is no need to change the original system codes. By using aspects, it was always possible to add new functionality to the previously developed distributed system without modifying the original codes. This approach adds modularity to systems behaviors and enhances the extendibility of the system.

This paper is structured as follows: In the following section, necessary background information is detailed. Section 3 depicts frameworks and design issues of the aspect oriented secure communication system. Performance evaluation results are depicted in Sect. 4, and finally, conclusion and future works are explained in Sect. 5.

2 Background

2.1 Aspect Oriented Programming

Aspect-Oriented Programming [7], is a relatively new complementary model to software design, and it enables the separation of crosscutting concerns by addressing it in a modular unit, called as *aspect*. In AOP, the system is divided into two parts: *the core system,* which contains the main functionalities and it is traditionally implemented using OOP methodology and *the aspect system,* which consists of the crosscutting functionalities, and it is implemented with helps of AOP methodology.

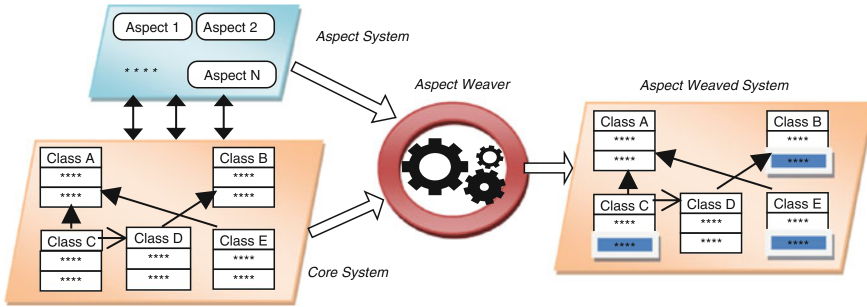

Fig. 1 Aspect oriented system design

At the programming level, an *aspect* is a software entity in which it modularizes the crosscutting concerns that are typically scattered across classes in the object-oriented system. An aspect definition contains two main elements: *pointcuts* and *advices*. An *advice* is a method that adds a behavior to an existing main system codes A pointcut is a description that specifies when, where and how to invoke the advice. A pointcut is conceptually defined predicate that is used to identify *join-points* in the system. A join point is a well-defined point in the execution of the program where additional behavior can be added. Finally, weaving is a composing process of transforming the system by linking core and crosscutting concerns together, thereby constructing an evaluated final system (as shown in Fig. 1).

2.2 Java RMI

A Java remote method invocation (RMI) takes the RPC concept towards distributed object communications in which client object locates the server object (possibly on a different machine) and remotely invoke its methods through the server's stub and skeleton functions. In traditional networking, it can be accomplished by writing an IP socket code to enable two objects on different machines send messages to each other. On the other hand, Java RMI lets us communicate these remote objects as if they are on the same machine.

The main Java RMI component consists of three elements: *Client, Server and Registry*. The client invokes a method on the remote object, the server object owns the remote object and processes the request, and registry works (in stub and skeleton) as a name server that enables objects to use unique names (Fig. 2).

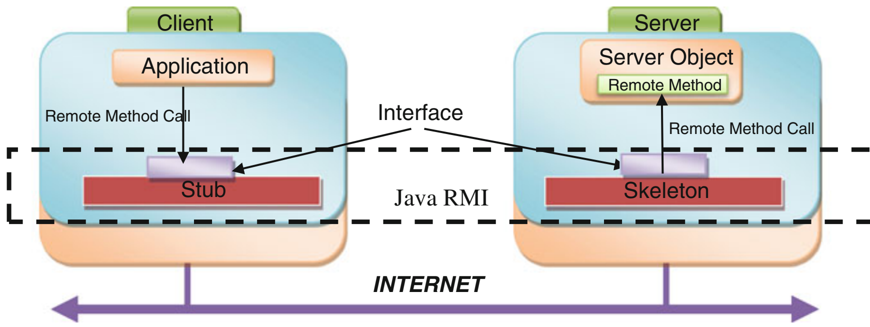

Fig. 2 Java RMI communication model

3 System Design

Most of the previously developed distributed systems did not address the secure communication issue that is a nonfunctional requirement in software engineering process. For continuing to use these types of unsecure systems in the real world, they should be enhanced by adding necessary reliable communication features. However, this is a challenging task to upgrade the application without corrupting the main business logic of the system. The aim of the secure communication is to protect messages from accidental or malicious modifications while they are in transit between two objects. This mechanism can be established by using an encrypted communication channel between these communicating objects.

As in information systems, encryption is an important method for implementing confidentiality in distributed systems. Unfortunately, applying effective encryption affects the functionality and performance of the system as a whole. Because security is migrated from the system security approach to the internal structure of the applications separately. As a result, encryption codes, as well as other security concerns, scatter in the main logic of each application, and it is a complicated task to implement this especially in large-scale systems. This crosscutting approach makes AOP be a potentially ideal candidate for implementation in distributed systems.

It is important for the application developers to decouple the security related modules from the primary system codes as much as possible. The use of aspects requires fewer changes, almost none, to this primary code and exhibits improved modularity over object-oriented implementation.

To develop secure communication systems preferred additional codes are based on common Java security packages and AspectJ [8] programming environment. It can be used as general-purpose AOP extension to Java and it enables modular implementation of components and crosscutting concerns on a single Java platform efficiently [9].

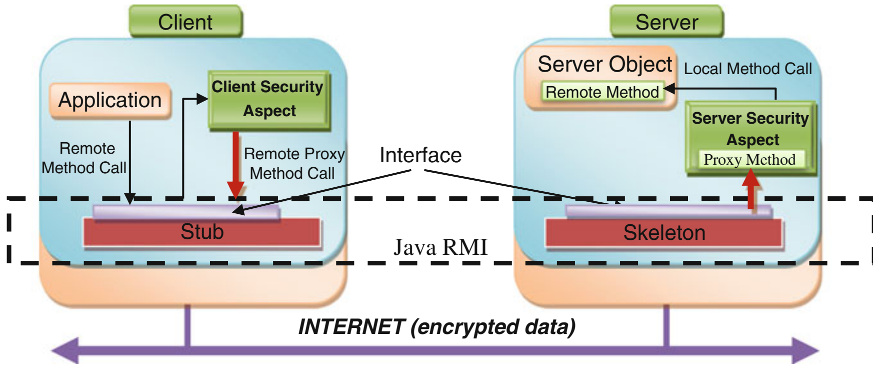

Fig. 3 Aspect oriented secure Java RMI communication model

To securely communicate a distributed system, which communicates with Java RMI as shown in Fig. 2, a security mechanism by using aspects and necessary proxy interfaces and methods is designed as shown in Fig. 3.

For enabling secure communication, message should be sent as encrypted to the server side and returned data from the method should also be encrypted. To achieve this, nine code changes are required to use Java RMI securely:

1. Remote server object must implement a proxy interface *(Object_Int)*, which extends the *Remote* interface, and all its methods must throw a *RemoteException*.

```
public interface Object_Int extends Remote {
public byte[] sendObj(byte[] e_data) throws RemoteException;
}
```

2. This proxy interface should have a proxy method that gets “byte[]” as a parameter and it should return byte[] as return type of the method. Because encrypted messages are transferred as byte[].
3. Necessary encryption and decryption algorithm methods *(`encrypt()` and `decrypt()`)*, which uses symmetric key, and a key storage mechanism should be implemented both in server side aspects and client side aspects.

```
public byte[] encrypt(byte[] data) {/*Encryption algorithms*/}
public byte[] decrypt(byte[] encrypteddata) {/*Decryption algo-
rithms*/}
public byte[] toByteArray (Object obj){/*Object to byte[] conver-
sion*/}
public Object toObject (byte[] bytes) {/*byte[] to Object conver-
sion*/}
```

4. For transferring all types of messages (containing simple parameters and each types of objects, which implemented *Serializable* interface), necessary *byte[] to Object(`toObject()`)* and *Object to byte[] (`toByteArray()`)* methods should be implemented both in server and client side.
5. Client objects remote method invocation command should be handled by an aspect (*Encrypt* aspect whose algorithm shown below) and converted to an encrypted bytes and after that by calling new proxy method this data should be sent to server side.
6. Server object cannot use this encrypted message. This message should be decrypted by the proxy method. Therefore, firstly, this incoming encrypted message is caught by the *Decrypt* aspect, and then, the decrypted message should forward to the remote server object.
7. After server object finished its run, the result message should be transferred in encrypted mode also. This also compensated by the *Decrypt* aspect. The decrypted data is taken from the server object, then it is encrypted according to the security mechanism, and finally this encrypted data is returned to the Client Object.
8. The incoming encrypted data is decrypted in the *Encrypt* aspect (in client side) according to the security mechanism. Obtained data is returned to Client Object.
9. Remote method calls must include a *try {…} catch* statement to deal with *RemoteExceptions*.

4 Performance Evaluation

An experiment platform is tested for a small scale Distributed Computing Platform, which runs on a local area network. Instead of a solution model like adding each encryption and decryption operations to these code places one-by-one, by using AOP approach, developing two necessary security aspects is sufficient and will be more secure/robust than the first solution approach. Test platform is developed on PCs, which are configured as in Table 1.

Table 1 Test platform properties

Properties	Values
CPU	Intel (R) Core(TM) i7-2630QM CPU @2.00 GHz.
Operating system	64 bit Windows 7 Ultimate edition
RAM	6.00 Gbyte
Message size	Min: 9 bytes- Max: 1008 bytes
Independent runs	5,000 times
LAN connection	1000 MByte ethernet
AspectJ version	1.6.12

Table 2 Performance comparison

Remote message sending time	Without aspects (ms)	With aspects (ms)[a]
Minimum	0.681	1.329
Maximum	1.269	24.491
Average	0.882	1.974

[a] Encryption time should be added to this time. It differs according to using encryption algorithm

Table 2 shows the composing a message and sending it to a Remote Object via Java RMI with and without AOP. Undoubtedly, adding these aspects slightly decreases the performance of the system with comparison to adding manually. Because, while developing these aspects, all exceptional situations are considered in the aspect codes, which cannot be executed in most of the joinpoints.

AOP is typically used in large scale and complex software development processes for meeting the some nonfunctional software requirements like modularization and quality of services. By using AOP, a large-scale distributed system could easily be converted to a secure communication in a modular way.

5 Conclusion

This paper presents how AOP approach can be used to resolve the tangled secure communication concern in a distributed system, which is developed with Java-based communication middleware (Java RMI). AspectJ is used as aspect oriented platform in a distributed system case study.

The communication related codes in distributed systems are spread across and are tangled into different classes or methods in the implementation phase. Therefore, it is difficult to modularize this concern in a separate functional module with Object Oriented Programming paradigm. Besides this, adding security concept to this tangled code structure is a challenging task. This type of spread codes and crosscutting concerns can be encapsulated into a modular unit, called as aspects, by using AOP. After that, secure communication can be enabled by modifying these aspects.

Secure communication aspects are weaved with java byte code without changing the original main application code. Advice codes of encryption and decryption mechanisms are applied in execution flow of the main program at join points when a match occurred with specified signature of aspects' pointcuts.

The main advantage of updating a distributed system with AOP approach is that there is no need to change the codes in the software. By this way, system security mechanism is modularly grouped in units/aspects also. If system developers want to change the security mechanism of this upgraded system, the only thing they need to do, is updating the necessary codes in these aspects.

At the same time, this paper shows separation of security concerns in a distributed application, which is one of the main non-functional requirements of a good software engineering approach. By this approach, some software quality factors like understandability, readability and modularity are also increased.

References

1. Yang, F., Aotani, T., Masuhara, H., Nielson, F., Nielson, H.R.: Combining static analysis and runtime checking in security aspects for distributed tuple spaces. In: Proceedings of the 13th İnternational Conference on Coordination Models and Languages (COORDINATION'11), Reykjavik, Iceland, pp. 202–218 (2011)
2. Stevenson, A., MacDonald, S.: Smart proxies in Java RMI with dynamic aspect-oriented programming. In: IEEE International Symposium on Parallel and Distributed Processing-IPDPS 2008, pp. 1–6 (2008)
3. Bostrom, G.: Database Encryption as an Aspect. In: Proceedings of the Workshop on AOSD Technology for Application-level Security, UK (2004)
4. Kotrappa, S., Kulkarni, P.J.: Multilevel security using Aspect oriented programming AspectJ. In: International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom), pp. 369–373 (2010)
5. Yang, F., Masuharab, H., Aotanib, T., Nielsona, F., Nielsona, H.R.: AspectKE*: Security Aspects with Program Analysis for Distributed Systems. In: Demonstration Track of the 9th International Conference on Aspect-Oriented Software Development (AOSD'10), Rennes and Saint Malo, France (2010)
6. Sahingoz, O.K.: Secure communication with aspect Oriented approach in distributed system programming. In: Academic IT Conference 2012—Usak, Turkey. 1–3 Feb 2012 (in Turkish)
7. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: Proceedings of the 11th European Conference on Object-Oriented Programming, pp 220—242 (1997)
8. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of AspectJ, in ECOOP 2001—Object-Oriented Programming 15th European Conference, pp. 327–353. Budapest Hungary, Springer (2001)
9. Toledo, R., Nunez, A., Tanter, E., Noye, J.: Aspectizing Java access control. IEEE Trans. Softw. Eng. **38**(1), 101–117 (2012)