國立交通大學

資訊科學系

碩 士 論 文

一 個 建 立 於 訊 息 導 向 中 介 軟 體 上 的 可 抽 換 式 安 全 架 構

A Pluggable Security Framework on Message Oriented Middleware

研 究 生：柯憲昌

指導教授：袁賢銘　教授

中 華 民 國 九 十 四 年 六 月

一個建立於訊息導向中介軟體上的可抽換式安全架構
A Pluggable Security Framework on Message Oriented Middleware

研　究　生：柯憲昌　　　Student：Shien-Chang Ko

指導教授：袁賢銘　　　Advisor：Shyan-Ming Yuan

國 立 交 通 大 學
資 訊 科 學系
碩 士 論 文

A Thesis
Submitted to Institute of Computer and Information Science
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer and Information Science

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

一個建立於訊息導向中介軟體上的可抽換式安全架構

研究生：柯憲昌　　　　　　指導教授：袁賢銘

國立交通大學資訊科學研究所

摘要

近年來，隨著網際網路的快速發展，訊息導向中介軟體（Message-Oriented Middleware）成為企業間傳遞訊息最普遍使用的工具，隨著這股熱潮，昇陽公司制定了 Java 訊息服務應用程式設計介面（JMS API），提供一個統一的標準介面，讓建立於訊息導向中介軟體之上的應用程式具有可移植性，而 Persistent Fast Java Messaging（PFJM）即是一套相容於 JMS API 的訊息導向中介軟體，並加強了其永續訊息與效能等特性。

隨著訊息導向中介軟體廣泛的應用於網際網路服務，以往被忽略的安全問題也慢慢浮出檯面，本篇論文將討論訊息導向中介軟體的安全議題，並於 PFJM 的基礎之上，提出一套可抽換式的安全架構，"可抽換" 意味著應用程式開發者只要透過簡單的修改設定檔，就能以不同的安全模組來建構其安全環境，而不須為了使用不同的安全架構，修改應用程式本身，透過抽換式的方法，應用程式開發者所使用的安全架構可隨其需求改變，提高開發應用程式的彈性。

A Pluggable Security Framework on Message Oriented Middleware

Student：Shien-Chang Ko   Advisor：Shyan-Ming Yuan

Department of Computer and Information Science
National Chiao Tung University

Abstract

With the rapidly growth of Internet, Message-oriented Middleware (MOM) had became the widespread used tool for delivering messages between companies. Sun Corporation had been aware the trend and defined a Java Message Service Application Programming Interface (JMS API) standard. This standard provides a set of uniform interface for application development and makes applications more portable. Persistent Fast Java Messaging (PFJM) is a JMS compliant Message-oriented Middleware and it has some outstanding features such as persistent message and high performance.

MOM had be widely adopted in Internet and the security issues not been noticed in the past had been discussed more and more. This paper will discuss the security issues on MOM. Based on PFJM, we bring up a pluggable security framework. "Pluggable" means application developers only need to modify configurations and plug in many different security modules to build a secure message delivering system. He/She needs not to modify applications in order to adopt different security strategies and be more flexible on developing.

Acknowledgements

首先要感謝在這兩年間指導我的指導教授 袁賢銘 教授，在論文指導上給了我很寶貴的建議並引導我找出論文方向，另外要感謝 蕭存喻 學長和 吳瑞祥 學長，在每次的小組會議上給我豐富的意見讓我可以撰寫出此篇論文，也謝謝小組裡的成員 沈上謙、葉倫武、顏志明，在研究上彼此交換意見、彼此學習，感謝上謙在我們的計畫中幫了很多忙，也恭喜小武順利當上博士生。

此外也要感謝系計中的 鄭建明 助教和 林佳慶 助教，在我寫論文的期間給我熱情的協助，傳授我寫論文的技巧以及幫我修改錯誤的英文。

最後要感謝我的父母 柯筵雄 和 吳疏美，有你們的栽培才能讓我考上交通大學，提供良好的環境讓我可以順利完成學業，謝謝您們對我的支持和鼓勵。

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Preface

With the rapidly growth of Internet at decades, the e-services environment had been gradually more practicable. In the past, information exchange between different departments of a corporation was based on physical paper. The communication between companies usually used FAX and telephone. Along with the appearance of many services on the Internet, E-Mail service substituted physical paper. The FAX telephone line was replaced by the Internet messages between companies. Internet had become the one unexpendable medium. On the one hand, companies wanted their services more easily accessible, and on the other, for saving their costs on handling the regular transactions, they built their own information system by their way. For example, the stock management system is used to manage the amount of products and automatically communicate to the providers via Internet messages when the amount of products is less than lower-bound. Internet greatly improved the productivity and convenience. However, building an information system is time consuming. Programmers must pay more attention on the details of network message delivery and system architecture. This disturbs programmers from business model and delays the time to market.

Afterward, businesses noticed the problem and found the necessary to adopt middleware to assist the development of information system. According to this reason, Message-oriented Middleware (MOM) was developed to help programmers for not to directly access the lower lever and disagreeable network protocols. MOMs are usually expressed by a set of readable APIs. Programmers can use the API to send and receive network messages and focus on designing business models. By this way, information system can quickly be available. However, when programmers move from one MOM system to another, the codes about communication in an application must be modified because every MOM's API is not the same. For portability, Sun defined a standard of Java Message Service Application Programming Interface (JMS API)[1] adopted by many MOM providers. The advantages of the standard interface are programs written by this API can run on many MOM products and programmers only learn one API.

JMS 1.1 standard was finalized at 2002. Nowadays MOM providers mostly support the JMS 1.1 standard, such as SonicMQ[2], FioranoMQ[3], OpenJMS[4], … etc. Persistent Fast Java Messaging (PFJM)[5] is a Message-oriented Middleware designed by our laboratory. In addition to support JMS 1.1 standard, we enhance the persistent message and high performance features for increasing the scalability greatly.

1.2 Motivation

Since the e-commerce was more and more popular, network security became an important issue. The messages delivered via network in plaintext may be snooped by crackers. So MOM providers began to add security functions to their products, including user authentication, trust authorization and message encryption. But the functions were usually and tightly bound with the implementation of MOM's core. Furthermore, the security functions were developed by MOM providers, which are preventing third-party providers from plugging in their security functions into MOMs, so this makes applications less flexible.

Therefore we bring up a pluggable security framework on MOM. The core concept is to adopt the loosely-coupled design. Based on this design, we can separate the MOM's core functionalities from security functionalities. The security functionalities can be packed in the form of modules. While developing, programmers only concern with the business models, not the security issues. During deployment, programmers can just simply modify the configuration and parameters, and use the suitable security modules to construct a secure message system.

Based on PFJM, we will realize the modular concept on security functionalities. In addition, we will implement several useful security modules to demonstrate the advantages of the pluggable framework.

1.3 Research Objectives

In this paper we discuss the security issues on MOM. The focus we emphasize is how to flexibly and easily add security functions into MOM, not the security functionalities itself. The implementations of security functions are out of scope in our discussion. We expect under our design, the flexibility of security functions of MOM can be as great as possible.

Flexibility

The more flexibility the middleware provides the more flexibility the application behaves. With the pluggable framework, programmers have the greatest flexibility using different authentication architectures, different authorization policies, and various encryption algorithms without modify any source codes.

Easy to use

The objective of MOM is to simplify the complicated message exchange protocols. Programmers can write simple codes for sending messages. The objective of security functions on MOM should be the same. With the pluggable framework, programmers can only modify configurations and add appropriate security functions into MOMs.

Scalability

Different applications have different security frameworks. For example, some small-scale systems need only simple authentication such as password, while other large-scale systems need more sophisticated authentication. With the pluggable framework, programmers have the ability to build different security framework.

1.4 Thesis Organization

This research is organized as following: in Chapter 2, the background information and related works are reviewed. We briefly introduce the concept of Message-oriented Middleware (MOM) and Java Message Service (JMS). We also take a more deeply introduction to our previous MOM implementation – Persistent Fast Java Messaging (PFJM) and indicate what security issues are related to MOM. Chapter 3 describes the design of pluggable framework and the related interfaces. Chapter 4 depicts how to integrate pluggable framework and Lightweight Directory Access Protocol (LDAP) to build a secure message delivery environment. Chapter 5 discusses many kinds of issue on design, including security, performance, resource management, etc. And last, in Chapter 6 there is a brief conclusion for our design and we give some ideas for the future works.

Chapter 2 Background & Related Works

This chapter describes MOM and related security issues to you. Section 2.1 introduces JMS and shows two different MOM architectures. Section 2.2 will introduce Persistent Fast Java Messaging designed by our laboratory and illustrates the features and relationship with JMS. Section 2.3 depicts other MOM products on market. Section 2.4 discusses the security issues on MOM and explains the design methodology of the products mentioned above. We will list a table for comparison and readability. Also we will indicate the shortcomings on the design of these products. Finally we introduce the Java Authentication and Authorization Service defined by Sun. Section 2.5 summarizes this chapter.

2.1 Java Message Service (JMS)

Java Message Service defined by Sun Corporation and other cooperators is a set of standard interfaces for message delivery. With JMS, programmers can create, send, receive, and read messages via the simple interfaces. The JMS standard only defines the semantic of interfaces, not how to implement. Every MOM provider can have their own implementation. Today almost every MOM products are compatible with JMS. The design architectures can be divided into two categories: central architecture and distributed architecture.

2.1.1 Central Architecture

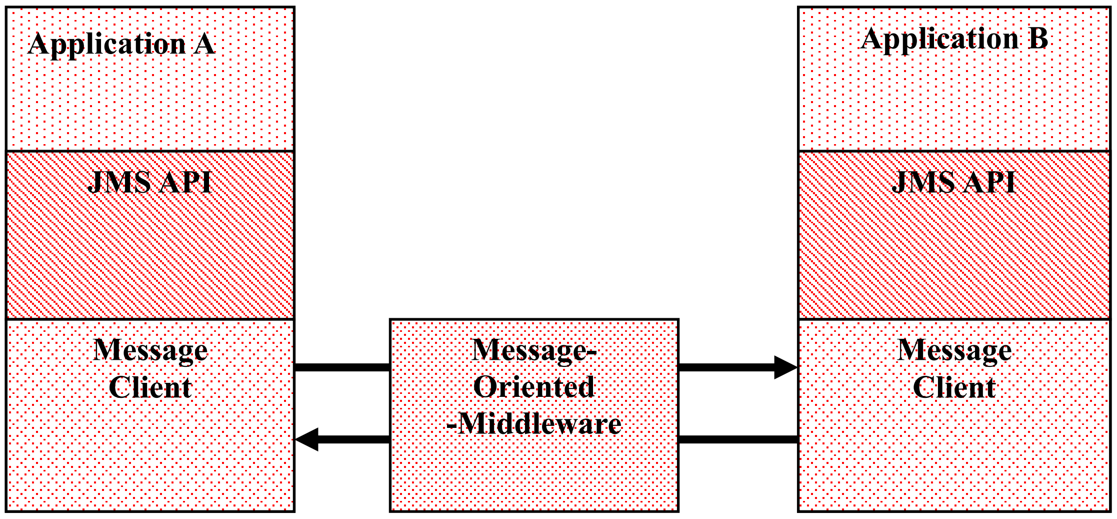


Figure 2-1 Central architecture of MOM design

Figure 2-1 is the central architecture of MOM design. A master server is responsible for message delivery and all applications are clients. When applications want to send messages out, it becomes sender, calling the JMS API and giving the destination of the messages. Messages will be delivered from application to central server. Central server will look for the destination of messages and send to the appropriate client. The destination of messages can be either Topic or Queue. Topic mode is a one-to-many mode. It means the sender can publish a message to Topic and receivers who subscribe the Topic will receive the message. While Queue mode is a one-to-one mode, the sender pushes messages into Queue and only one receiver can get back the message.

The shortcoming of central architecture is the server-bottleneck problem. If the central server gets low performance or even becomes failure, the overall message exchange system will becomes unavailable. However, it has the advantages of easy management and uncomplicated design.

2.1.2 Distributed Architecture

Another design methodology is the distributed architecture. Under this architecture, there is no master server and jobs of message delivery are distributed to every client. Figure 2-2 depicts the architecture. Because there is no longer a server, every client must be aware of some information of other clients, for example, IP and Port. The advantage of distributed architecture is the loading of original central server is divided and distributed to every client. So the single-point-failure problem does not exist. On the contrary, resource management will be complicated and difficult.

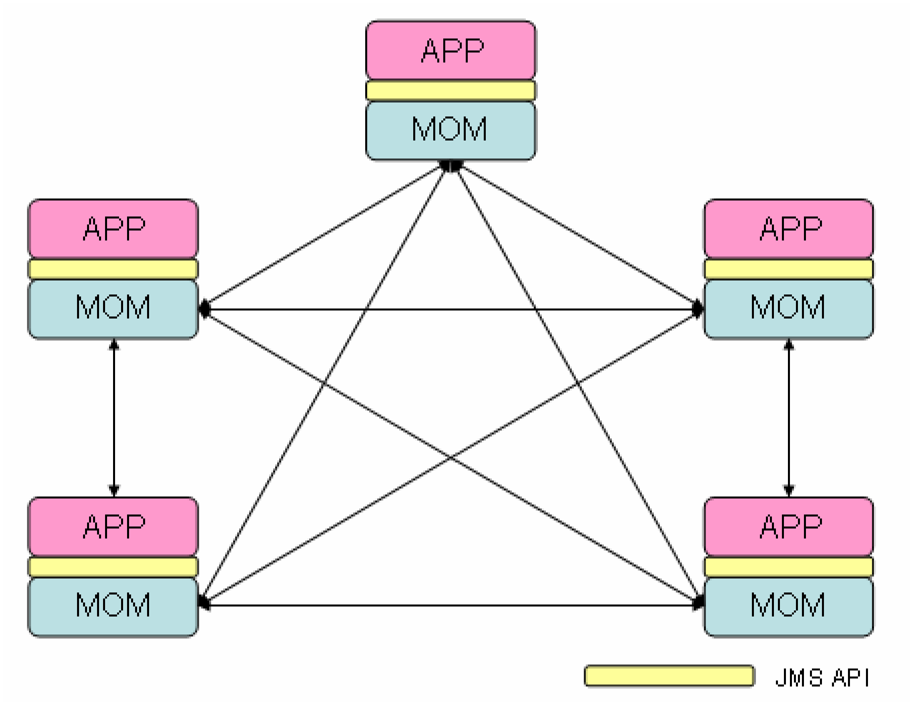


Figure 2-2 Distributed architecture of MOM design

2.2 Persistent Fast Java Messaging (PFJM)

Persistent Fast Java Messaging is a JMS compliant product designed by our laboratory. PFJM adopts the distributed architecture and implements the message delivery protocol using IP multicast technology. Figure 2-3 illustrates the overall architecture of PFJM.

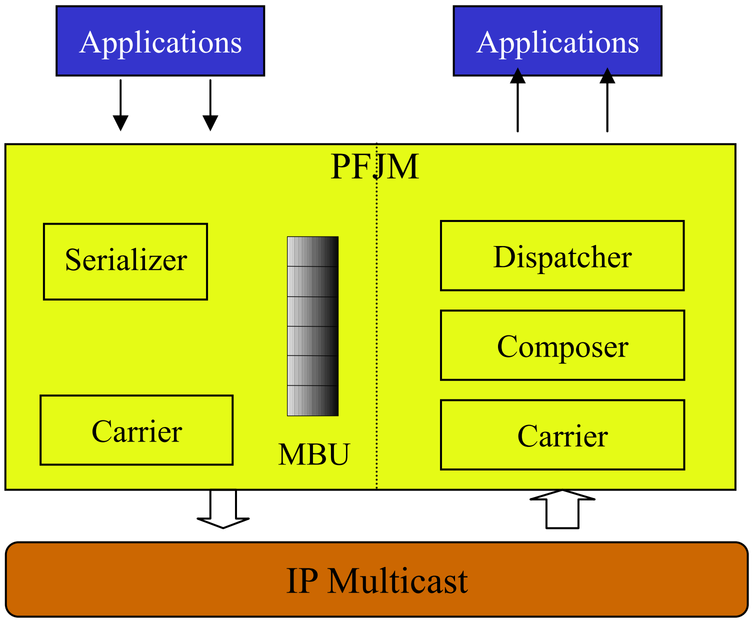


Figure 2-3 Architecture of PFJM

Every PFJM instances will simultaneously be client as well as server. When one PFJM instance starts, it connects to other PFJM instances via multicast messages in the network and then obtains necessary information. When delivering message, application talks to PFJM and indicates what message to send and which Topic the message should go. PFJM is responsible for dividing the message into many Memory Buffer Units (MBUs) that have same size and put them into the delivery buffer in memory. Then Carrier will pick up these MBUs and actually send them out via multicast. While PFJM is a role of receiver, it will register the appropriate multicast channel which corresponding to Topic. Incoming messages (MBUs) will be received by Carrier and then be composed by Composer. Finally the origin message is delivered to application.

In addition to following JMS standard, PFJM emphasizes on some features such as persistent message and high performance.

2.2.1 Persistent Messaging

PFJM adopts the distributed architecture and avoids single-point-failure. It reduces the lose rate of message. In addition, PFJM implements the persistent message feature using file system. When PFJM receives the messages delivered from application, it makes a copy to the file system. If system failure or any irregular error while messages are still in memory, the messages do actually lose but will be recovered from file system and redelivered when PFJM is restarted in the future. Figure 2-4 illustrates the concept of persistent message.

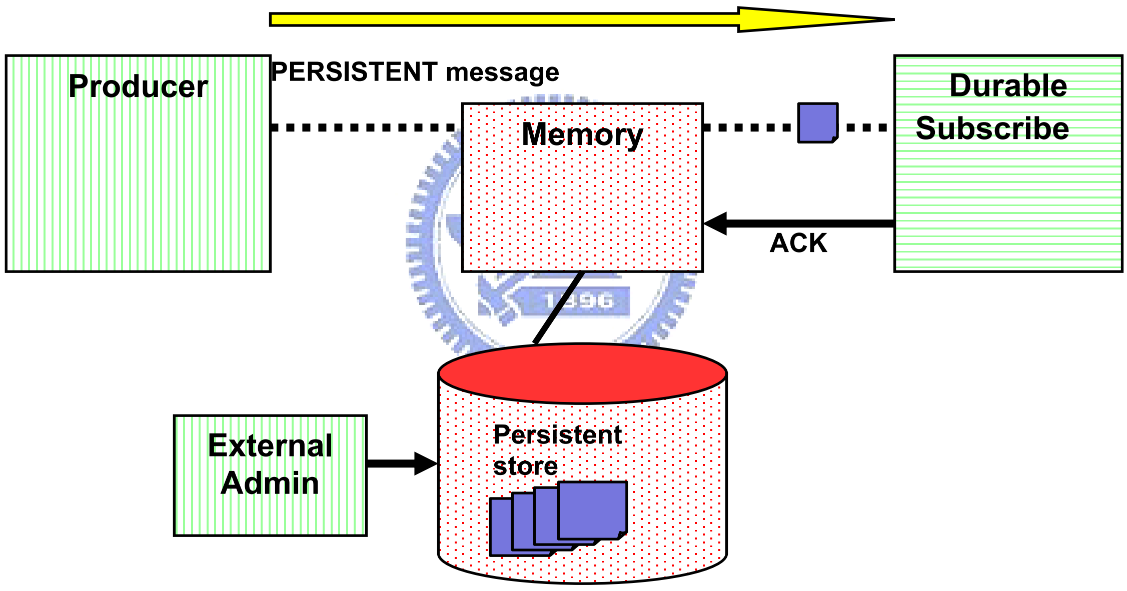


Figure 2-4 Persistent messaging of PFJM

2.2.2 Performance

There are two main classifications of messages in PFJM, one is the control message and the other is the actual message. There is a fixed multicast channel for control message. As PFJM starts, it communicates with other PFJM instances via control messages. When delivering messages, the actual messages will be sent out via multicast. By using the IP multicast technology, we have the advantages of high scalability and high performance. Because when the number of subscriber increases, publisher still needs to send message once. Obviously this design will have more performance than traditional centralized MOM. Furthermore, we adopt the NAK[5] technology to accomplish flow control and retransmission instead of heavy protocol such as TCP. The overhead introduced by protocol can be reduced.

2.3 Other Products

There are some other similar MOM products such as Sonic Software's SonicMQ[2], Fiorano Software's FioranoMQ[3] and OpenJMS[4] which is released in the form of open source. They are all designed as central architecture.

OpenJMS

OpenJMS provides the most basic MOM functionalities. It supports both peer-to-peer mode and publish/subscribe mode and uses JDBC for persistent messages.

SonicMQ

In addition to basic functions, SonicMQ emphasizes its cluster feature. Although SonicMQ is the central architecture, it supports multiple servers for constructing a SonicMQ cluster. With the feature, the loading can be distributed to every server and the server-bottleneck is no longer a problem. Besides, the amount of publishers and subscribers can be larger.

FioranoMQ

FioranoMQ emphasizes its high performance. The official website of Fiorano

Software indicates its FioranoMQ version 8.0 is more effective than SonicMQ version 6.0 about 2 to 10 times. In addition, FioranoMQ supports transformation between XML documents and JMS messages.

2.4 Security

2.4.1 Security Issues

Network security becomes more and more important. Performance is not only the focus of MOM design. Something likes privacies, non-replacement, non-denial are more concerned. It means every publisher and subscriber in the message exchange environment needs to be authenticated and the messages transferred in the network need to be encrypted first. So MOM needs some functions such as user authentication, authorization and message encryption. We noticed that MOM providers took no concern on security issues in the past and recently they add these functionalities enthusiastically

The following table is the comparison of security functionalities of SonicMQ, FioranoMQ and OpenJMS.

Table 2-1 Security comparison of SonicMQ, FioranoMQ, OpenJMS

Issues \ Products	SonicMQ	FioranoMQ	OpenJMS
Authentication	ACL、Password-based	ACL、Java Realms	Password-based
Authorization	ACL	ACL	None
Encryption (Message)	RSA B-safe、pluggable cipher suites	Pluggable cipher suites	None
Encryption (Channel)	HTTPS、SSL	HTTPS、SSL	SSL

According to table 2-1, we can see both SonicMQ and FioranoMQ use pluggable cipher suites, but all of them use static authentication and authorization. It is possible to carry out using central architecture but when adopting distributed environment, it has the problems of hard deployment and hard management. We need to design a more flexible authentication and authorization strategies.

2.4.2 Pluggable Cipher Suite

The pluggable cipher suite SonicMQ and FioranoMQ used is based on the Cipher interface of JCE (Java Cryptography Extension)[6]. Its advantage is encryption algorithms can be dynamically added into Java environment. In the Sun's implementation, it supports AES, Blowfish, DES series, and RSA series algorithms.

2.4.3 Java Authentication and Authorization Services (JAAS)

Because more and more applications need authentication and authorization functions, Sun defined a standard – Java Authentication and Authorization Services[7] for applications to build user based access control easily. JAAS adopts a model of Pluggable Authentication Module (PAM)[8]. As figure 2-5, LoginContext API is called by application and it utilizes the LoginModule SPI (Service Provider Interface) as a bridge with various login modules. We only modify configurations and use different authentication strategies. JAAS's authorization portion adopts the Security Manager and Policy provided by Java2 to control access permissions.

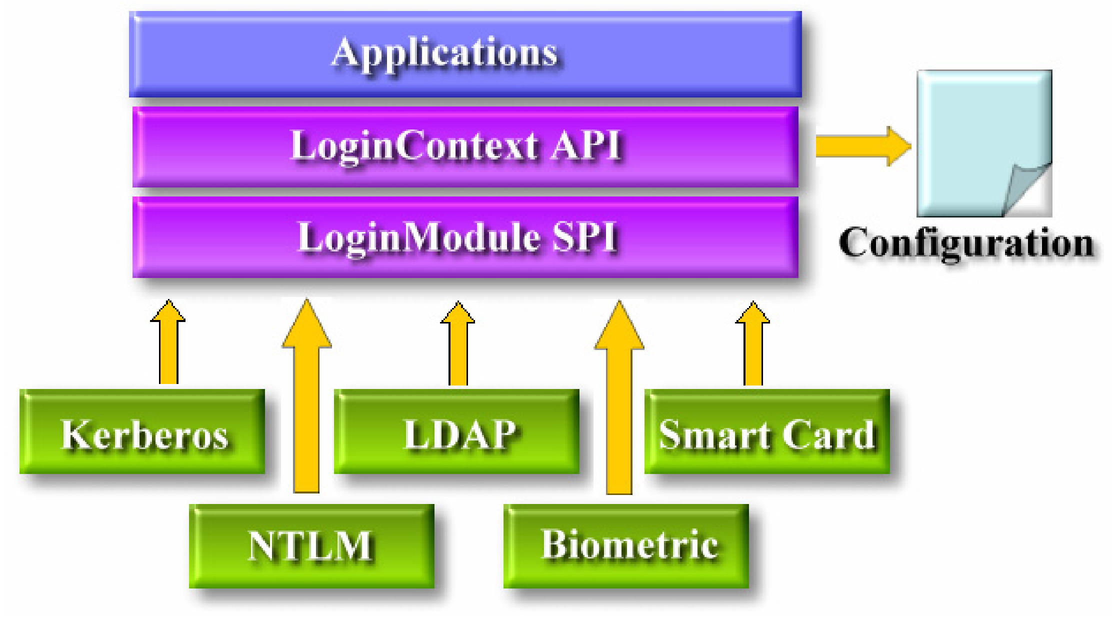


Figure 2-5 The authentication portion of JAAS

Although JAAS makes programmers more easily to integrate authentication and authorization functionalities with their programs, but it still has some disadvantages. First, the configuration of authentication and authorization is separated into two files, and are not easy to read. Second, the authorization rules are stored at local file system, it has some risks to be accessed by hackers. Third, if we apply JAAS to distributed systems, the management of authorization rules becomes difficult. So we need to improve these shortcomings of JAAS.

2.5 Summary

In this chapter we compare the security functions between MOM products on market. We observed every MOM product adopts a fixed manner for authentication and authorization. But under distributed environment, the variation of architecture is large and resources management is difficult. We need a more flexible way to accomplish authentication and authorization. JAAS is a framework defined by Sun for flexible authentication. But it has some disadvantages in authorization and can't be applied to distributed systems. So we bring up a more flexible framework for security.

Chapter 3　Pluggable Framework

This chapter describes the pluggable framework. Section 3.1 introduces the concept of the pluggable framework. Section 3.2 explains how can we adopt the pluggable framework into PFJM and make it more flexible. Section 3.3 shows the design of interfaces of the pluggable framework. Section 3.4 introduces the design of configurations.

3.1 Concept of Pluggable Framework

At the early stages, programmers usually wrote their codes in the function-oriented model. They first of all wrote the core functionalities and then appended other secondary functionalities. However their design is hard coded core and secondary functionalities. By the way, modifying either the core functionalities or secondary functionalities will influence each other. And programmers must be careful when they did something with the codes to avoid mistakes and bugs. Subsequently, programmers noticed the problem and adopted the modular design model. It means after core functionalities were built, other functionalities can be added into the software in the form of modules, without modifying the software.

The design methodology of modular model requires full plan at the design phase. The interfaces of modules need to be defined first. We call these interfaces the Service Provider Interface (SPI). Once interfaces are built, programmers only need to follow these interfaces and carry out their implementation without changing the origin codes.

The biggest benefit of modular design is the division and collaboration between software components are more clear. Every function can have several different implementations and we only need to change the configuration to use another implementation. It makes the software system more loosely-coupled.

3.2 Architecture

We observed that the functionalities of MOM can be categorized into several main classifications: message delivery, security, message filtering, logging, monitoring, persistent message and the lowest layer, protocol binding. We can design these functionalities with pluggable modules.

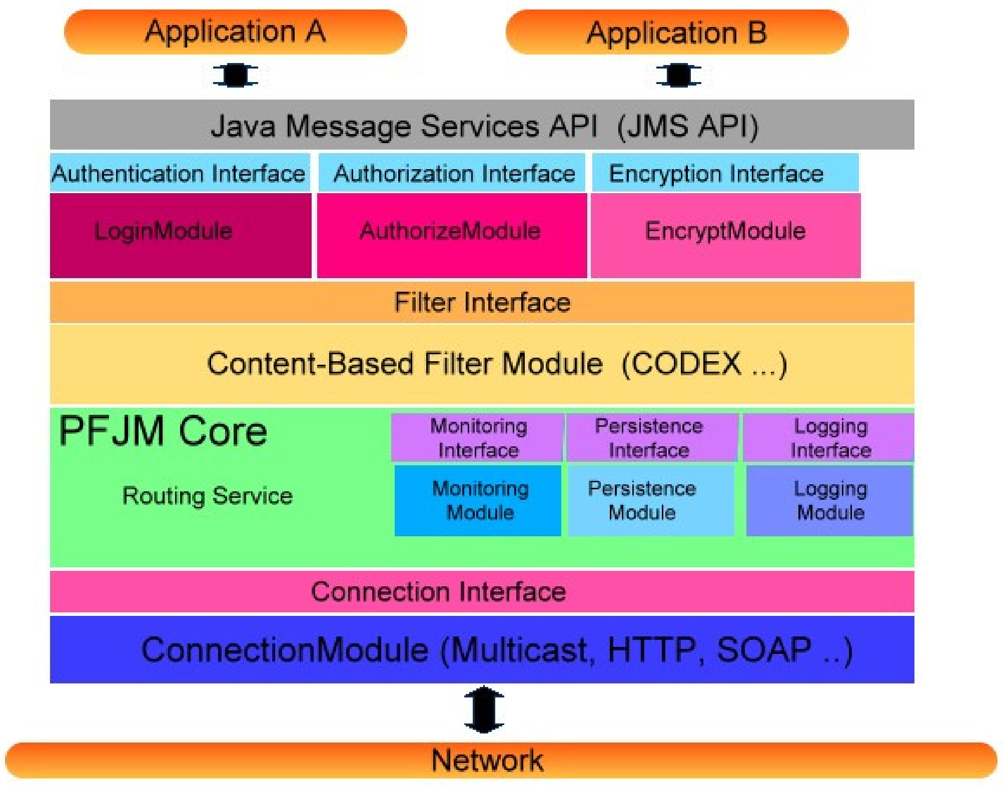


Figure 3-1 Architecture of pluggable framework on PFJM

Figure 3-1 is the architecture designed with the pluggable framework. Under the JMS

API, there are authentication interface, authorization interface and encryption interface and all of them talk to related modules to actually perform authentication, authorization and encryption. Next part is the filter interface, as well as persistent message interface, logging interface, and connection interface.

3.2.1 Security Interfaces

At figure 3-1, the authentication, authorization, and encryption interfaces are security related interfaces. An abridged MOM system should have authentication functionality. When a client accesses the MOM system, it should be authenticated and be a legal user of the system. There are various authentication methods such as Kerberos[9], NTLM[10] or LDAP[11], …etc. The authenticator can be the MOM server or any third-party authentication server. In our design, we can plug in any authentication modules such as Kerberos module, NTLM module or LDAP module to accomplish the authentication task.

After authentication successes, the MOM system may authorize clients. It determines which the Topic client can create or remove, as well as if they can publish to or subscriber from topic. Authorization usually executes with authentication. Similar to authentication, authorization can be performed on an authentication server or another authorization server.

When a client passes the processes of authentication and authorization, and it has the appropriate access right, it can start publish or subscribe messages. In order to protect the message against any stealing in network, the published message should be encrypted. There are many algorithms for encryption, like DES[12], AES[13], …etc.

We can adopt appropriate algorithms by changing the encryption module.

3.2.2 Filter Interface

The JMS standard has defined most essential application based filter. The client as a receiver can indicate which message it interests according to the information in the message headers and message properties. But it is too fundamental for complicated applications. Some MOM providers may add more powerful filters such as content based filters to improve their quality.

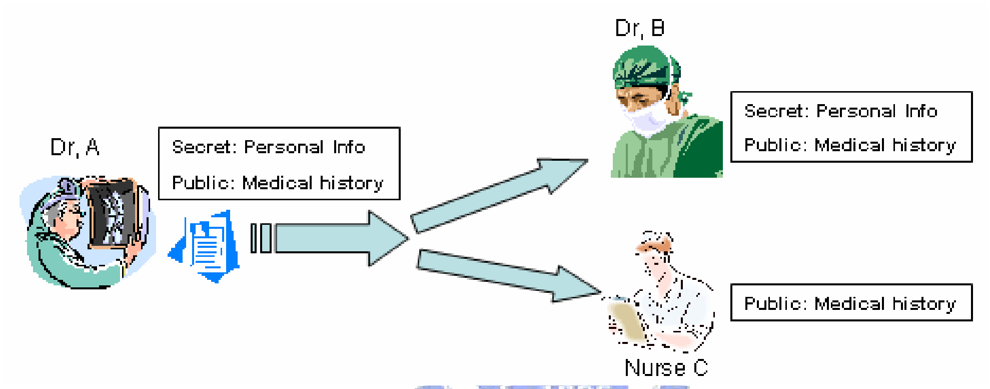


Figure 3-2 An application of message filter

Another kind of filter is authentication based filter. For example, figure 3-2 is a hospital information system built with MOM. While Dr. A publishes an anamnesis to Dr. B and Nurse C, Nurse C can only read the public section of the anamnesis because she is authenticated to be a general user that has lower permission.

3.2.3 Other Interfaces

Other interfaces are monitoring interface, persistent message interface, logging interface and connection interface. Monitoring interface provides third-party companies the opportunity of applying their monitor products into MOM. Persistence interface provides a method for messages to be stored in the file system or database.

Logging information can be stored to file system or sent to logging daemon via logging interface. At last, connection interface provides the chance of using many different protocol bindings.

3.3 Interface Design

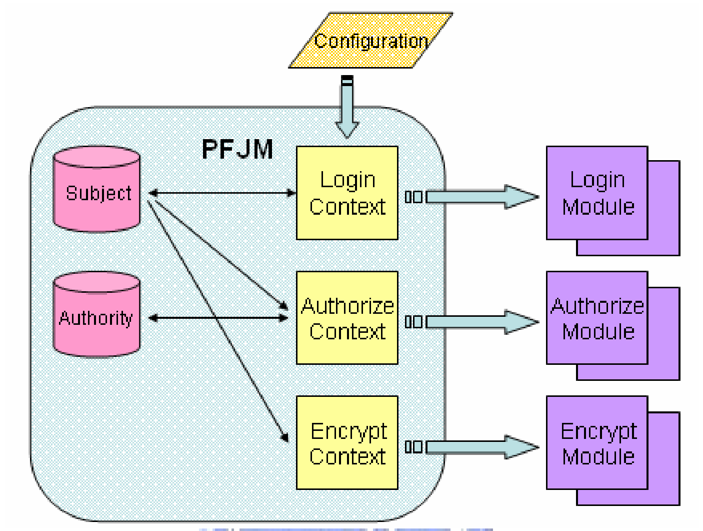


Figure 3-3 Implementation details

In this paper we will introduce the design of authentication, authorization and encryption interface. Figure 3-3 is the implementation details. The principals of our design are simplified interfaces and are compatible with the authentication modules of JAAS.

When PFJM instance is executed, it will read the configuration and determines which modules are loaded. Then it manages these modules via various contexts. LoginContext is responsible for managing LoginModules. The number of LoginModule can be zero or more. It means there are many processes to complete authentication. If authenticated, LoginContext will put some confidentiality into the Subject data structure. AuthorizeContext is responsible for managing

AuthorizeModules to authorize PFJM instance. AuthorizeModule gains the access right according to the confidentiality in the Subject data structure and stores the right to the Authority data structure. When PFJM needs to publish or subscribe messages, the Authority will be check first to determine whether it has the appropriate rights. EncryptContext is responsible for managing EncryptModules. EncryptModule is used to encrypt and decrypt messages. EncryptModule may get the private key form Subject.

3.3.1 Authentication Interface

In order to be compatible with JAAS's authentication modules, we adopt the authentication interface of JAAS and implement our own LoginContext to manage them. Figure 3-4 is the LoginModule interface.

```
public interface LoginModule
{
        boolean abort ();
        boolean commit ();
        void initialize (Subject subject, CallbackHandler callbackHandler,
                                    java.util.Map sharedState, java.util.Map options);
        boolean login ();
        boolean logout ();
}
```

Figure 3-4 LoginModule interface

The state diagram of interface call is as figure 3-5.

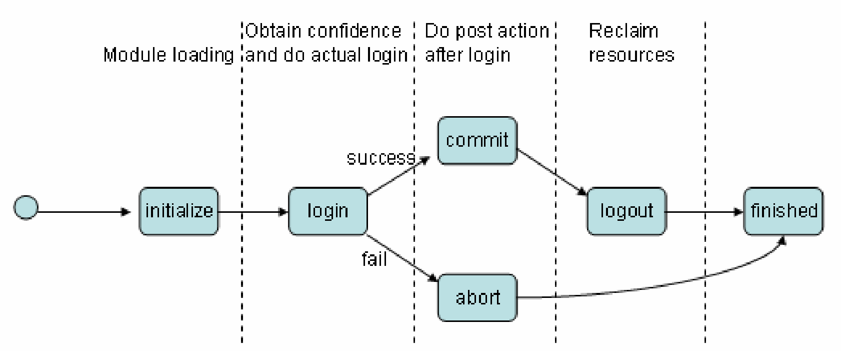


Figure 3-5 State diagram of authentication interface

Initialize

When PFJM starts, LoginContext will request every LoginModule's initialize function to construct and load the module. It will also pass in a Subject data structure for LoginModules to store the generated confidentiality after authenticated. When login processes are actually performed, CallbackHandler is necessary for LoginModules to get required information from user, such as user name and password. The sharedState options are useful when two or more LoginModules need to communicate. The options which named option are parameters that would be passed into LoginModules.

login

As all LoginModules are initialized, LoginContext will invoke the login function of every module. Login method is the place that actually performs authentication. Different LoginModules will have different implementations of login. No matter the result of authentication, LoginModule should memorize its authentication state and return the result to LoginContext. LoginContext will decide whether the overall authentication passed or not after all login processes are performed. Overall authentication is determined by the significant value of every module which comes from configuration. The significant value can be "required", "sufficient", "requisite" or “optional”. “Required” means this module is required and if any required modules failed, the overall authentication will fail. But the login processes behind this module will still be performed. “Sufficient” means if the present module successes, the overall authentication will be passed at the premise of no “required” or “requisite” fails and the later on login processes will be skipped. “Requisite” is the same with “required”. The difference between these two options is when requisite modules are failed, the later processes are skipped. “Optional” means whatever the result of authentication is; it has no effect on overall authentication.

The following figure is an example of authentication status. There are four authentication modules: SampleLoginModule, NTLMLoginModule, SmartCard, and Kerberos modules. The significant values are respectively “required”, “sufficient”, “requisite” and “optional”.

SampleLoginModule	required	pass	pass	pass	pass	fail	fail	fail	fail
NTLoginModule	sufficient	pass	fail	fail	fail	pass	fail	fail	fail
SmartCard	requisite	*	pass	pass	fail	*	pass	pass	fail
Kerberos	optional	*	pass	fail	*	*	pass	fail	*
Overall Authentication		pass	pass	pass	fail	fail	fail	fail	fail

Figure 3.6 Authentication status[7]

commit or abort

If overall authentication passed, LoginContext will call every module’s commit function. Commit function needs to determine if its login process is passed. If so, it will put some confidentiality into Subject, and clean the private data that user inputs. If overall authentication failed, LoginContext will call all every module’s abort method to clean up the privacy.

logout

When authentication process is finished, LoginContext will call every module's logout method and clear the Subject data structure.

3.3.2 Authorization Interface

Because the authorization of JAAS utilizes the Java2 security model, it leads to some problems described above in distributed system. We redesign this portion using the pluggable concept by defining authorization interface Figure 3-7 is the outline of authorization interface.

```
public interface AuthorizeModule {
   public void initialize (Subject subject, Authority authority, Map
sharedState,Map options);
   public boolean retrievePolicy ();
   public boolean cleanup ();
}
```

Figure 3-7 Authorization interface

Figure 3-8 is the state diagram of authorization interface.

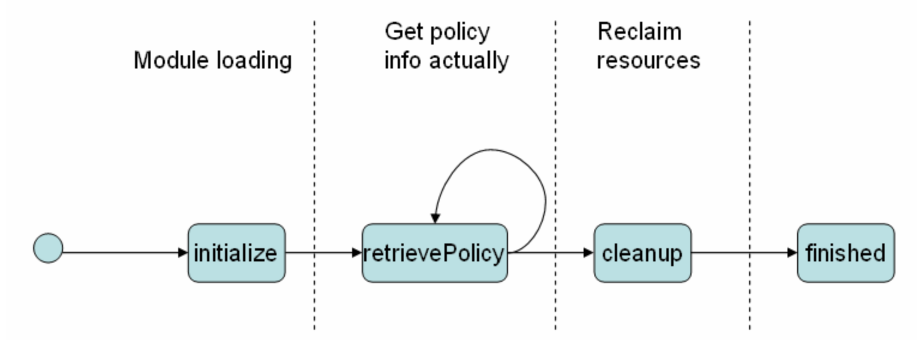


Figure 3-8 State diagram of authorization interface

initialize

After authenticated, LoginContext will give the control to AuthorizeContext. As the same as LoginModule, AuthorizeContext will call every AuthorizeModule's initialize method and pass in the Subject which gained from LoginContext. Moreover, AuthorizeContext will pass in an Authority data structure for storing permission information. The sharedState and options parameter is the same as LoginModule.

retrievePolicy

After initialized, AuthorizeContext will call every module's retrievePolicy method. This method should retrieve permission information according to Subject. It may get information from the local or remote authorization server. The permission information should be put into Authority.

Because we may add new permission information during the development, we design the Authority data structure as flexible as possible. And we classify permissions into categories. Figure 3-9 depicts the Authority data structure.

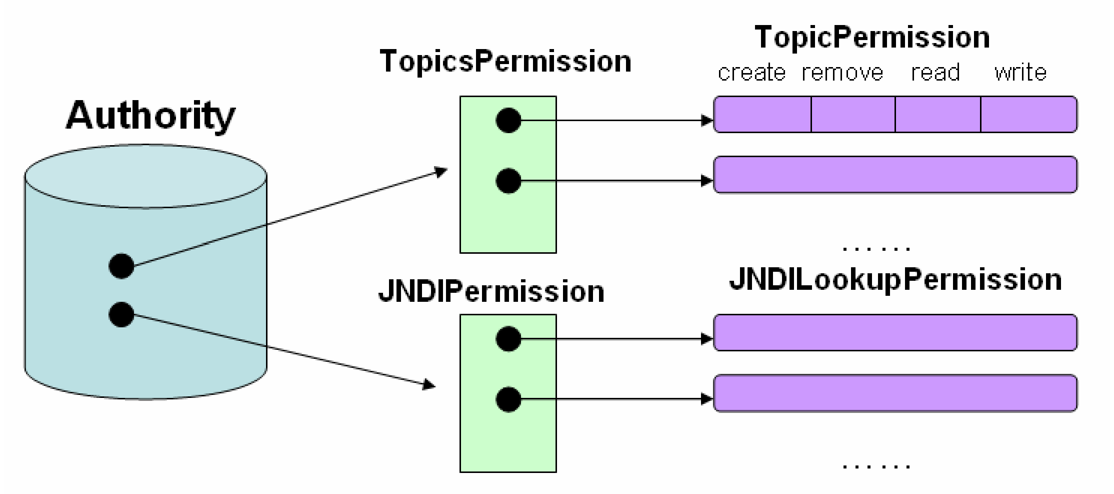


Figure 3-9 Data structure of Authority

cleanup

When PFJM terminated, AuthorizeContext will call the cleanup method of every module. This function should clear permission information stored in Authority.

3.3.3 Encryption Interface

```
public interface EncryptModule {
   public void initialize (Subject subject, Map sharedState, Map options);
   public byte[] getEncryptedMessage (byte[] msg) throws EncryptException;
   public byte[] getDecryptedMessage (byte[] msg) throws EncryptException;
   public boolean cleanup ();
   public boolean negotiate ();
}
```

Figure 3-10 Encryption interface

Figure 3-10 is the state diagram of encryption interface.

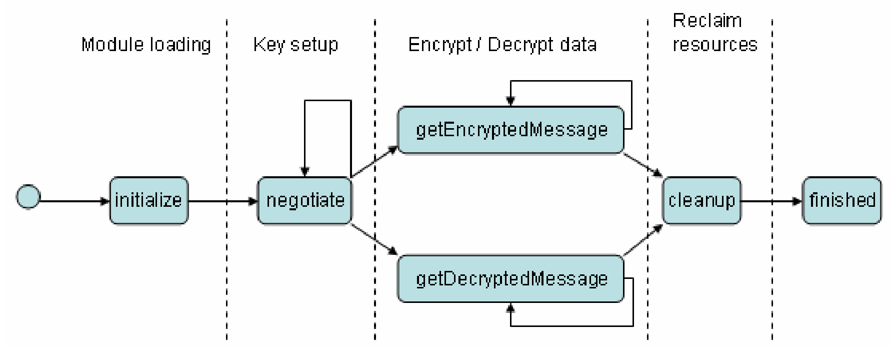


Figure 3-11 State diagram of encryption interface

initialize & negotiate

There may be many methods to do encryption. Some algorithms may be safer, but slower, and some are opposite. Via unified interface, we can plug in appropriate encryption modules at different circumstances. As the same as authentication and authorization, EncryptContext will initiate the module via calling the initialize method. Then it calls the negotiate method. This method must finish key setup.

getEncryptedMessage & getDecryptedMessage

When sending or receiving messages, EncryptContext will call the getEncryptedMessage to encrypt messages or getDecryptedMessage to decrypt messages.

cleanup

When PFJM terminated, EncryptContext will call this method to cleanup private data such as private key.

3.4 Configuration File

```
<Security>
  <!-- This part specifies which LoginModule should be invoked -->
  <!-- "needs" option can be 'required','sufficient','requisite','optional' -->
  <LoginModule name="com.cmc.jms.security.modules.LDAPLoginModule"
needs="required">
    <!-- You can optionally indicate which CallbackHandler should be used -->
    <CallbackHandler name="com.cmc.jms.examples.MyCallbackHandler"/>
    <Option debug="true"/>
    <Option server="140.113.88.237"/>
    <SharedState myState1="ok"/>
  </LoginModule>

  <!-- This part specifies which AuthorizeModule should be invoked -->
  <AuthorizeModule name="com.cmc.jms.security.modules.LDAPAuthorizeModule">
    <Option debug="true"/>
    <SharedState myState2="ok"/>
  </AuthorizeModule>

  <!-- This part specifies which EncryptModule should be invoked -->
  <EncryptModule name="com.cmc.jms.security.modules.DESEncryptModule">
    <Option debug="false"/>
    <SharedState myState3="no"/>
  </EncryptModule>
</Security>
```

Figure 3-12 Security configuration

The configuration file of PFJM is in the form of XML[14]. The Security section in this file is the security related setting. Figure 3-11 is an example; PFJM instance uses LDAPLoginModule to login the message exchange system. The server option informs LDAPLoginModule where the LDAP server is. Then it use LDAPAuthorizeModule to gain its permission and finally, DESEncryptModule to encrypt the outgoing message using Data Encryption Standard algorithm.

Chapter 4 A Security System on PFJM

After PFJM has integrated with the pluggable framework, we can adopt many security modules to construct a secure message exchange environment. In this chapter, we use LDAP as the basic building block.

4.1 System Architecture

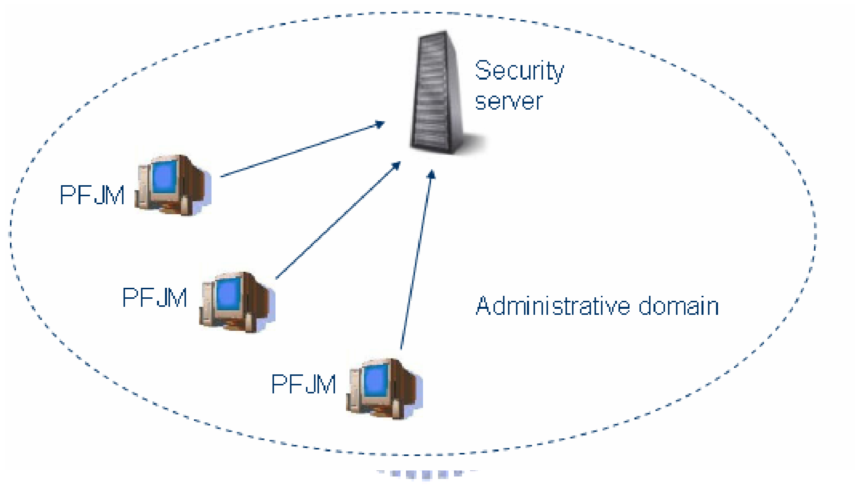


Figure 4-1 A secure system architecture for PFJM environment

In order to integrate authentication and authorization into PFJM, it is necessary to make a good plan to manage various resources such as user list and topic directory. Unfortunately, it is difficult to manage resources in the distributed system because the management protocol may be too complicated and less effective. Due to the difficulty and disadvantages, we use the centralized architecture to construct the security environment. Figure 4-1 is the outline of the architecture. There is a security server in PFJM environment for authentication and authorization. The advantage of this architecture is easy management. Besides, the confidentialities are stored in the security server, it is not a problem for local steals.

4.2 Security Server

We use LDAP (Lightweight Directory Access Protocol) as the security server. LDAP is a lightweight protocol for rapidly search for specific data in LDAP server. The data structure in LDAP server is tree architecture. Figure 4-2 is the data structure we stored in LDAP directory.

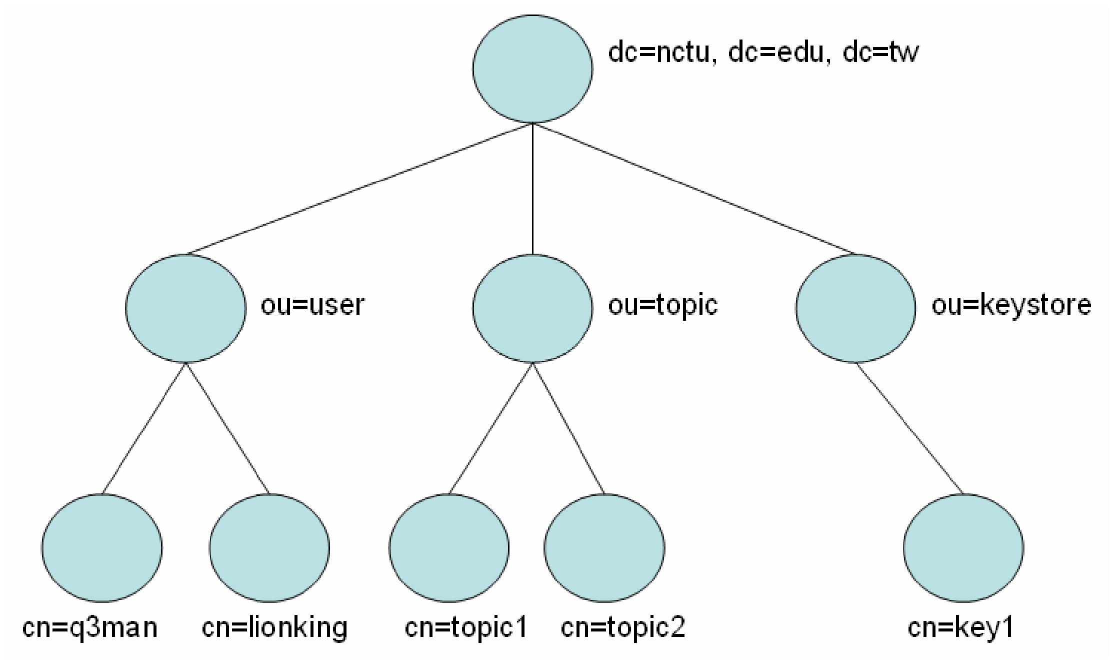


Figure 4-2 A example of LDAP directory

In this figure every circle is a node and composed by one or more properties. A property is composed by a couple of name and value. Every node has at least one index property, for example, cn=q3man. Index property is used to indicate the relative distinguished name (RDN). The path from the root of the tree structure to the specific RDN is called distinguished name (DN). For example, the left lowest node has the DN of cn=q3man, ou=user, dc=nctu, dc=edu, dc=tw.

We classify resources into three parts – user directory, topic directory and keystore directory. User directory is a list of user information such as user name and password. Topic directory has some topic information. Keystore directory is a place where private keys store.

We utilize OpenLDAP[15] server as the security server. OpenLDAP server is released in the form of open source. It provides complete LDAP services. It also has access control mechanism to restrict information that PFJM can access.

4.3 Modules Design

We use LDAPLoginModule to authenticate PFJM to the OpenLDAP server and authorize via LDAPAuthorizeModule. Finally we use DESEncryptModule to encrypt messages that sent to topic. The flow path is as figure 4-3.

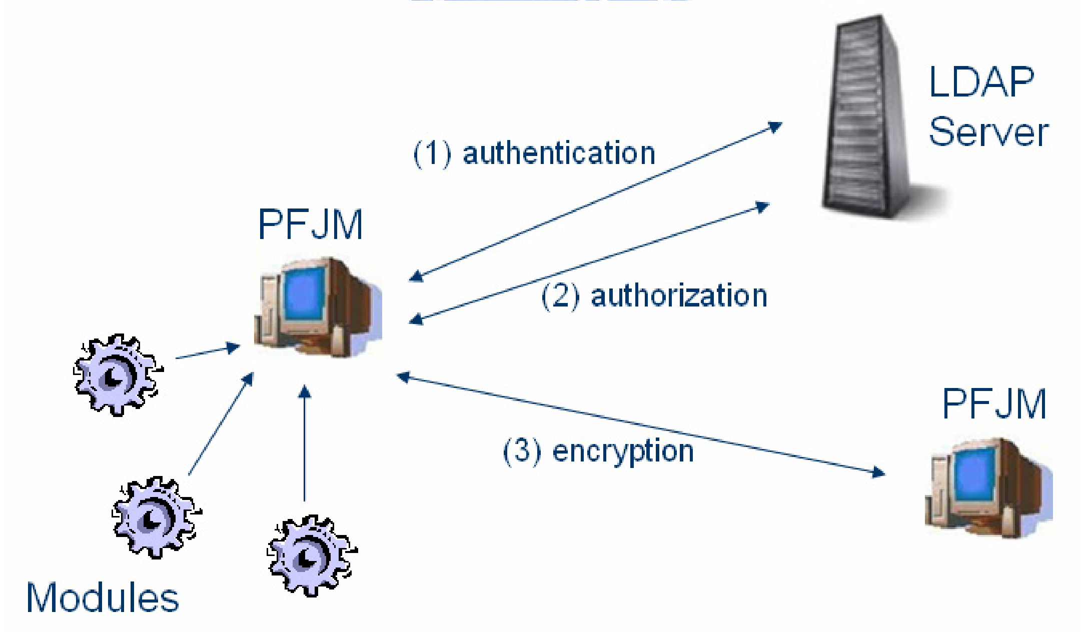


Figure 4-3 A PFJM environment with AAE

4.3.1 LDAP Authentication Module

LDAPLoginModule is responsible for authentication. It implements the LDAP protocol and communicates with the OpenLDAP server. LDAPLoginModule will ask for user name and password using CallbackHandler. Then it passes this information to OpenLDAP using SSL secure connection.

4.3.2 LDAP Authorization Module

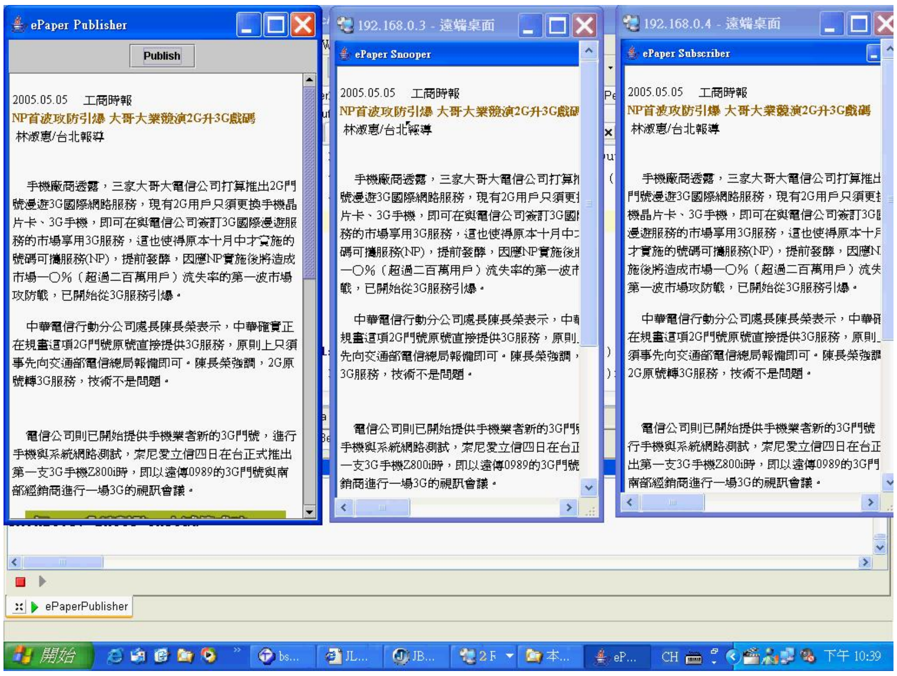
After LDAPLoginModule passed, LDAPAuthorizeModule is used to retrieve necessary permissions and pass to PFJM. If PFJM has no permission to perform an action, it returns exception.

4.3.3 DES Encryption Module

Finally the DESEncryption is used to encrypt messages. The module use 64 bits key space and Data Encryption Standard algorithm.

4.4 A Scenario

We setup a OpenLDAP server and write a simple e-Paper publishing system to demonstrate the advantages of the pluggable framework. As figure 4-4, the left window is an e-Paper publisher and the right window is an e-Paper subscriber. The middle window is an illegal snooper that eavesdropping the traffic between publisher and subscriber. With no security mechanisms, the snooper will successfully read the traffic.



2005.05.05　工商時報
NP首波攻防引爆 大哥大業競演2G升3G戲碼
林淑惠/台北報導

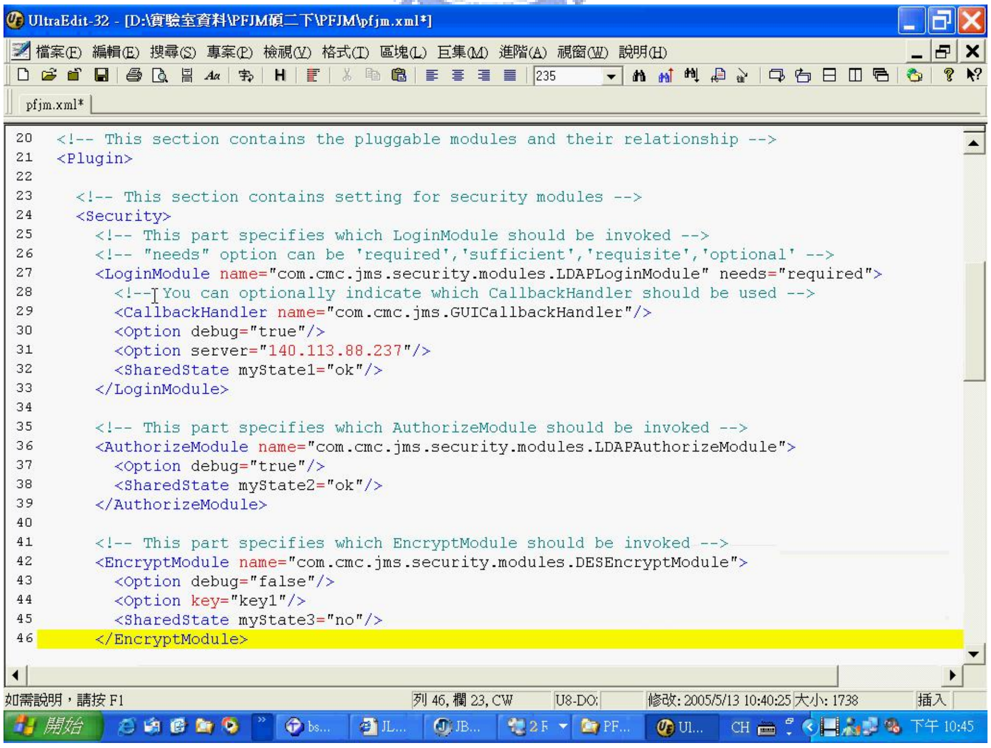
手機廠商透露，三家大哥大電信公司打算推出2G門號漫遊3G國際網路服務，現有2G用戶只須更換手機晶片卡、3G手機，即可在與電信公司簽訂3G國際漫遊服務的市場享用3G服務，這也使得原本十月中才實施的號碼可攜服務(NP)，提前發酵，因應NP實施後將造成市場一〇%（超過二百萬用戶）流失率的第一波市場攻防戰，已開始從3G服務引爆。

中華電信行動分公司處長陳長榮表示，中華確實正在規畫這項2G門號原號直接提供3G服務，原則上只須事先向交通部電信總局報備即可。陳長榮強調，2G原號轉3G服務，技術不是問題。

電信公司則已開始提供手機業者新的3G門號，進行手機與系統網路測試，索尼愛立信四日在台正式推出第一支3G手機Z800i時，即以遠傳0989的3G門號與南部經銷商進行一場3G的視訊會議。

Figure 4-4 e-Paper system (step 1)

In order to avoid eavesdropping against snooper, we plugged in LDAPLoginModule, LDAPAuthorizeModule, and DESEncryptModule at publisher and subscriber, as figure 4-5.

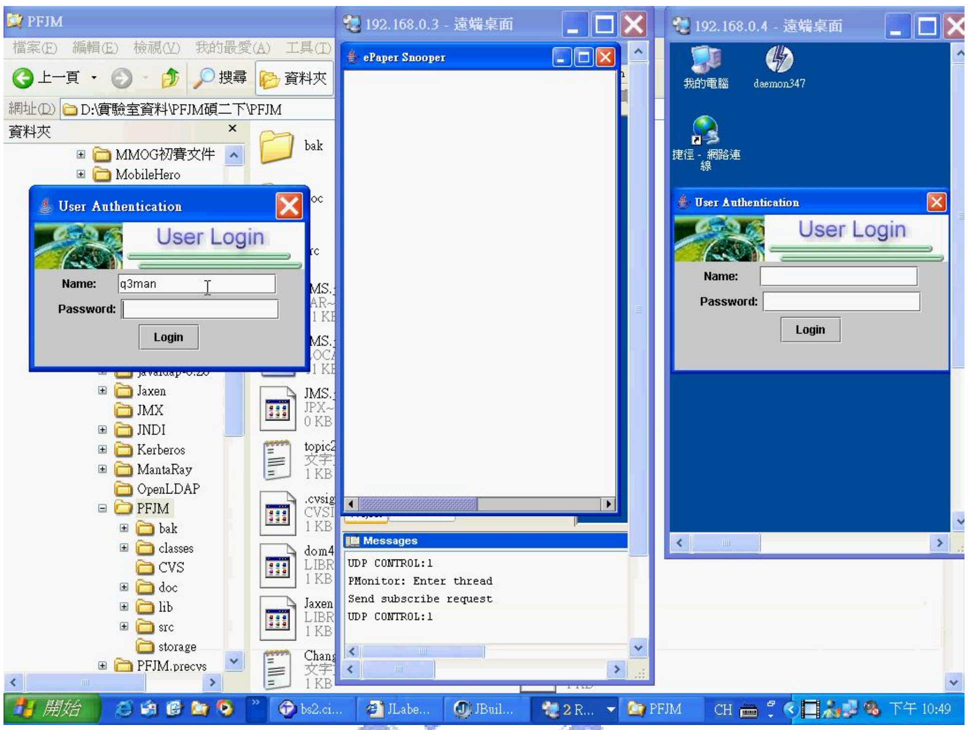


UltraEdit-32 - [D:\實驗室資料\PFJM碩二下\PFJM\pfjm.xml*]
檔案(F) 編輯(E) 搜尋(S) 專案(P) 檢視(V) 格式(T) 區塊(L) 巨集(M) 進階(A) 視窗(W) 說明(H)
pfjm.xml*
如需說明，請按 F1
列 46, 欄 23, CW
修改: 2005/5/13 10:40:25 大小: 1738
插入

```
<!-- This section contains the pluggable modules and their relationship -->
<Plugin>

  <!-- This section contains setting for security modules -->
  <Security>
    <!-- This part specifies which LoginModule should be invoked -->
    <!-- "needs" option can be 'required','sufficient','requisite','optional' -->
    <LoginModule name="com.cmc.jms.security.modules.LDAPLoginModule" needs="required">
      <!-- You can optionally indicate which CallbackHandler should be used -->
      <CallbackHandler name="com.cmc.jms.GUICallbackHandler"/>
      <Option debug="true"/>
      <Option server="140.113.88.237"/>
      <SharedState myState1="ok"/>
    </LoginModule>

    <!-- This part specifies which AuthorizeModule should be invoked -->
    <AuthorizeModule name="com.cmc.jms.security.modules.LDAPAuthorizeModule">
      <Option debug="true"/>
      <SharedState myState2="ok"/>
    </AuthorizeModule>

    <!-- This part specifies which EncryptModule should be invoked -->
    <EncryptModule name="com.cmc.jms.security.modules.DESEncryptModule">
      <Option debug="false"/>
      <Option key="key1"/>
      <SharedState myState3="no"/>
    </EncryptModule>
```

Figure 4-5 e-Paper system (step 2)

In figure 4-5, we indicate the LDAP server's IP and which key is used to encrypt messages. We use the GUICallbackHandler to ask for user name and password. Figure 4-6 is the looks of GUICallbackHandler.

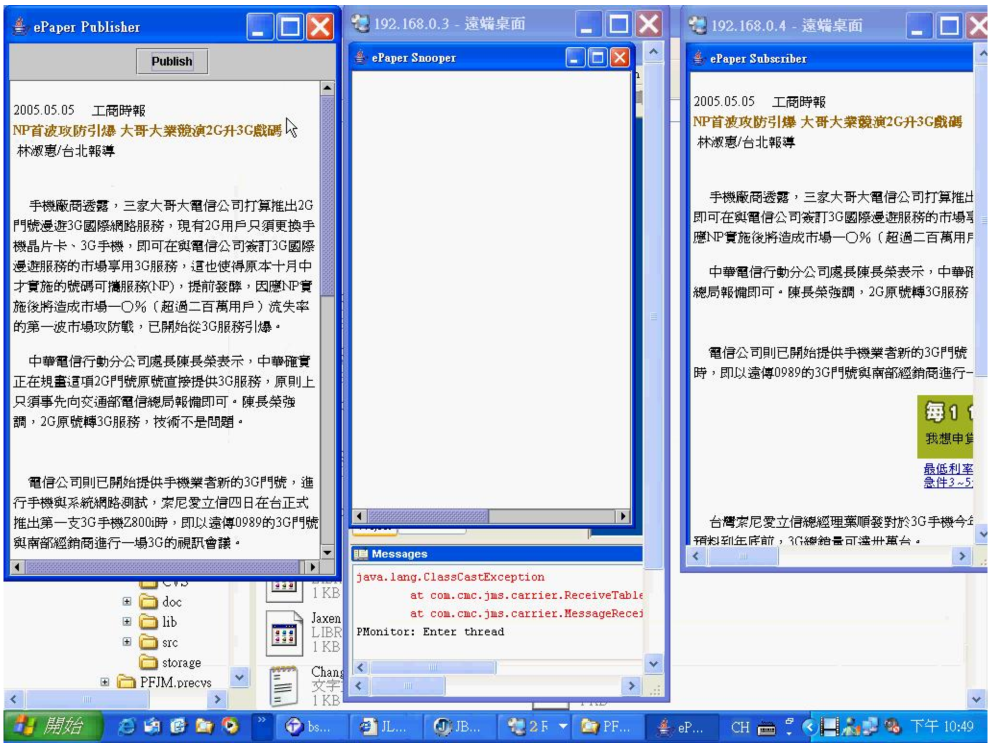


Name: q3man

```
UDP CONTROL:1
PMonitor: Enter thread
Send subscribe request
UDP CONTROL:1
```

Figure 4-6 e-Paper system (step 3)

Finally we can observe the snooper is not authenticated and can't get the private key for encryption. It can't receive the message that the publisher sent and get an exception, as figure 4-7.



2005.05.05　工商時報
NP首波攻防引爆 大哥大業競演2G升3G戲碼
林淑惠/台北報導

手機廠商透露，三家大哥大電信公司打算推出2G門號漫遊3G國際網路服務，現有2G用戶只須更換手機晶片卡、3G手機，即可在與電信公司簽訂3G國際漫遊服務的市場享用3G服務，這也使得原本十月中才實施的號碼可攜服務(NP)，提前發酵，因應NP實施後將造成市場一○%（超過二百萬用戶）流失率的第一波市場攻防戰，已開始從3G服務引爆。

中華電信行動分公司處長陳長榮表示，中華確實正在規畫這項2G門號原號直接提供3G服務，原則上只須事先向交通部電信總局報備即可。陳長榮強調，2G原號轉3G服務，技術不是問題。

電信公司則已開始提供手機業者新的3G門號，進行手機與系統網路測試，索尼愛立信四日在台正式推出第一支3G手機Z800i時，即以遠傳0989的3G門號與南部經銷商進行一場3G的視訊會議。

```
java.lang.ClassCastException
        at com.cmc.jms.carrier.ReceiveTable
        at com.cmc.jms.carrier.MessageRecei
PMonitor: Enter thread
```

2005.05.05　工商時報
NP首波攻防引爆 大哥大業競演2G升3G戲碼
林淑惠/台北報導

手機廠商透露，三家大哥大電信公司打算推出
即可在與電信公司簽訂3G國際漫遊服務的市場
應NP實施後將造成市場一○%（超過二百萬用

中華電信行動分公司處長陳長榮表示，中華
總局報備即可。陳長榮強調，2G原號轉3G服務

電信公司則已開始提供手機業者新的3G門號
時，即以遠傳0989的3G門號與南部經銷商進行

台灣索尼愛立信總經理葉順發對於3G手機今
預料到年底前，3G總銷量可達卅萬台。

Figure 4-7 e-Paper system (step 4)

From the above scenario, we illustrate the flexible and easy to use advantages of the pluggable framework.

Chapter 5 Discussion

In this chapter we discuss some ideas and trade-off during design of the pluggable security framework and describe the advantages and disadvantages.

5.1 Multiple Interfaces

In our design we define multiple interfaces for different functionalities instead of single interface. The main reason is different functionalities have dissimilar input and output. It means different parameters and return values of interfaces. Also, adapting single interface needs assistance of meta-data, for example, the deployment descriptor of EJB[16]. It will lead to performance waste. The following table lists the comparison of multiple interfaces and single interface designs.

Table 5-1 Comparison of multiple interfaces and single interface

features	Multiple Interfaces	Single Interface
Performance	High	Low
Readability	High	Low
Maintenance	Hard	Easy
Flexibility	Low	High
Deployment	Easy	Hard

Performance

Due to the input/output of multiple functionalities are different, the design of single interface needs some help of meta-data. For example, we can design a interface called

UniInterface：

```
public interface UniInterface
{
        public int preAction (DSPara in, DSRtn out);
        public int inAction (DSPara in, DSRtn out);
        public int postAction (DSPara in, DSRtn out);
}
```

Figure 5-1 A example of single interface

The data type DSPara is a flexible data structure for storing parameters. As the same, the DSRtn is a data structure for storing return values of the functions. The retrun value of every function states the execution state. At this scenario, every module must inherit this interface and the DSPara and DSRtn may be described as XML.

```
<DSPara>
        <Integer>500</Integer>
        <Byte>0x01</Byte>
        … …
<DSPara>
```

Figure 5-2 A example of parameters described in XML form

This is a key point that leads to performance waste. Because PFJM kernel needs to parse the XML first and reconstruct the DSPara and DSRtn data structure, it is time consuming.

Readability

As figure 5-1, we can’t quickly find out the semantics according to the UniInterface’s member functions. In our multiple interfaces design, the semantics is clear. For example, getEncryptedMessage means message encryption.

Deployment

The difficulty of deployment is also an important concern. The single interface design is more complicated on deployment because the types of parameters and return values are not fixed and also the semantics of functions are not clear. So the MOM's manager usually got confusion with deployment. In our design, the structure and semantics of configuration are clearer for easy deployment.

The advantage of using multiple interfaces is the semantics of interfaces is clear. Programmers can easily read and understand the interfaces. Unlike EJB, the categories of functions of MOMs are not infinite. So the design of multiple interfaces is possible.

Because there is only a set of interface, programmers only need be familiar with the interface. It is more convenient for maintenance using single interface. By the help of deployment configurations, it is also more flexible than multiple interfaces. But for clear semantic and easy deployment, we adopt the design methodology of multiple interfaces.

5.2 Middleware-layer Security

The reason why we bring security functions into middleware layer, not leave to application layer is to simplify the programming processes and programmers can focus on business model. Without security related codes, applications are more portable. Another reason is if middleware supports many security modules, programmers can easily built their security framework.

5.3 Performance vs. Security

It is usually a trade-off between performance and security. More safely encryption usually needs more CPU resources. By pluggable security framework, programmers can choice appropriate modules to fit their requirement. But under some circumstance, programmers have no idea about how safely the security module is and how much CPU resources the module needs. We bring up the classification mechanism and document it to programming guide. Figure 5-1 is an example.

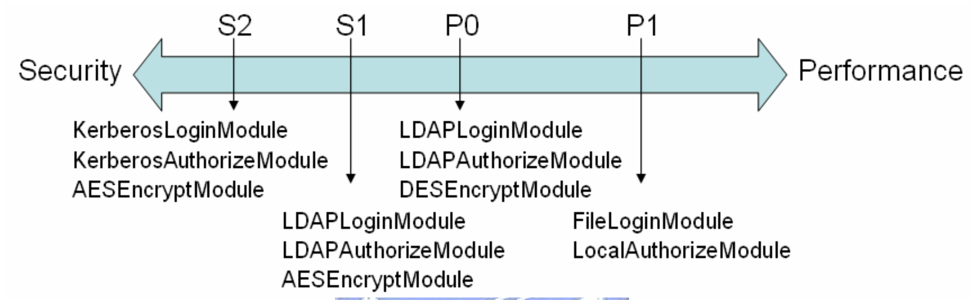


Figure 5-3 Trade-off between security and performance

In the example we classify modules into four levels. Level P0 uses LDAP for authentication and authorization, and DES for encryption. It provides medium security and performance. It is a guideline for programmers.

5.4 Distributed Management

It is very difficult in distributed management. The lookup, synchronization, addition, and deletion of resource are sophisticated enough for writing another paper. But under some circumstance it is indeed necessary. With pluggable security framework, third-party providers can plug in their distributed modules and use distributed security architecture with PFJM.

Chapter 6 Conclusion and Future Works

6.1 Conclusion

In the e-services world, MOMs are widespread used at various systems. Programmers can build their systems quickly by MOMs. MOMs simplify the sophisticated processes for delivering messages. For portability, Sun defined the JMS API and programs written using JMS API have the benefit of “write once, run anywhere”. The JMS API is a set of simplified API that programmers can learn easily. But Sun doesn’t define security related functionalities in JMS API. Nowadays, security issues are more and more concerned. So every MOM providers usually add their own security functions into their products.

In this paper, we discuss many security issues including user authentication, trust authorization, and message encryption on MOMs. We surveyed the methods how the MOM products on market solve the issues. And we conclude that the design is less flexible and can’t be adapted to distributed systems. So we bring up a pluggable security framework with more flexibility. We design our security functions in the modular concept and define several interfaces for security modules. Based on PFJM, we built the pluggable security framework into PFJM and write several security modules for it. Programmers can adapt to different security strategies by modifying configurations. In this paper, we designed a system that using LDAP server for authentication and authorization to demonstrate the flexibility of pluggable security framework.

The pluggable concept can be applied to other functions such as logging. By pluggable framework, the distinction and collaboration of components of software are clearer. And the maintenance and addition of the software are more convenient.

6.2 Future Works

There are many security related problems in MOMs, not just authentication, authorization and encryption. Other problems such as how to filter messages, how to manage topics are also important and complicated. Following we give some ideas for filtering and topic related issues.

Authentication-based Filter

We can add filter interface into PFJM for filtering messages. PFJM already has basic application-based filter. "Application-based" means the filter's rules come from applications. Another kind of filter is authentication-based filter. "Authentication-based" means its rules come from the identity of authenticated user. For example, the filter rules of nurse C in figure 3-2 filter the private portion of an anamnesis.

If application-based filter are used with authentication-based filter, the filter processes are as following.

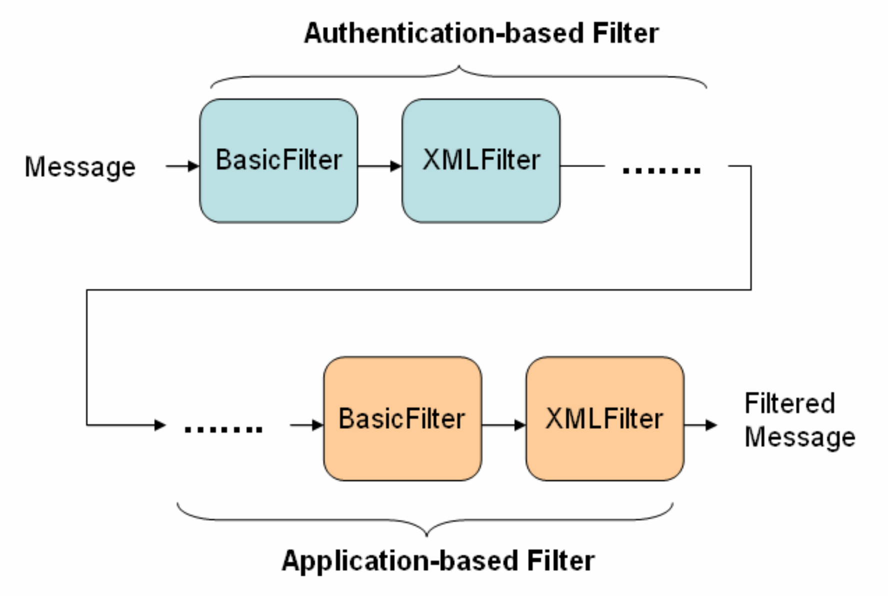


Figure 6-1 Combined usage of authentication-based and application-based filters

After authenticated, authentication-based filters will be executed. In the example, there are BasicFilter and XMLFilter. Application-based filters are executed according to the requirement of applications.

Pluggable Protocol

As figure 2-1, message encryption can be classified to per-message encryption and per-channel encryption. In our pluggable security framework, there is per-message encryption. By adding connection interface in figure 3-1, we can have the pluggable per-channel encryption such as HTTP, SSL, SOAP, SSH, etc.

Per-Topic Encryption

In our framework, all messages are processed by the same encryption modules. But more complicated, we expect messages are processed by different modules according to the topics. So every topic uses different encryption algorithms. A cracker who have successfully snooped one topic will have no idea to snoop another one. The PFJM environment is more secure.

Bibliography

[1] Sun Microsystem, Java Message Service Specification Version 1.1, April 2002

[2] Sonic Software’s SonicMQ, http://www.sonicsoftware.com/index.ssp

[3] Fiorano Software's FioranoMQ,

http://www.fiorano.com/products/fmq/overview.htm

[4] Project OpenJMS, http://openjms.sourceforge.net/

[5] Yu-Fang Huang, Tsun-Yu Hsiao, Shyan-Ming Yuan. A Java Message Service with Persistent Message, Proceeding of Symposium on Digital Life and Internet Technologies 2003

[6] Sun Microsystem, Java Cryptography Extension (JCE) Version 1.1

[7] Sun Microsystem, Java Authentication and Authorization Services (JAAS) Version 1.0, December 1999

[8] DEC-RFC 86.0 from SunSoft, Unified Login with Pluggable Authentication Modules (PAM), October 1995

[9] MIT, Kerberos: The Network Authentication Protocol

[10] Eric Glass, The NTLM Authentication Protocol, 2003

[11] RFC 3377, Lightweight Directory Access Protocol (v3)：Technical Specification, September 2002

[12] National Bureau of Standards, “Data Encryption Standard,” U.S. Department of Commerce, FIPS pub. 46, Jan. 1997

[13] National Institute of Standards and Technology (NIST), “Advanced Encryption

Standard (AES)”, FIPS Publication 197, Nov. 2001,

http://csrc.nist.gov/encryption/aes/ index.html

[14] Project dom4j, http://dom4j.org/

[15] Project OpenLDAP, http://www.openldap.org/

[16] Sun Microsystem, Enterprise JavaBeans Technology (EJB) Specification Version 2.1