國立交通大學

資訊工程系

碩士論文

CCL OSA:以 CORBA 為基礎的開放式服務存取平台

CCL OSA: A CORBA-based Open Service Access System

研 究 生：周家銘

指導教授：林一平　教授

中 華 民 國 九 十 四 年 六 月

CCL OSA:以 CORBA 為基礎的開放式服務存取平台

CCL OSA: A CORBA-based Open Service Access System

研 究 生：周家銘　　Student：Chia-Ming Chou

指導教授：林一平 博士　　Advisor：Yi-Bing Lin

國 立 交 通 大 學

資 訊 工 程 系

碩 士 論 文

A Thesis

Submitted to Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science and Information Engineering

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

CCL OSA:以 CORBA 為基礎的開放式服務存取平台

學生: 周家銘　　指導教授: 林一平 博士

國立交通大學資訊工程學系碩士班

摘　　要

Open Service Access (OSA) 是開放式的服務存取平台。該平台提供業者建立與部署兼具彈性與效率的行動服務。OSA 可以讓網路的營運業者，透過第三方應用程式與服務的提供者，來增加他們的收入。在 OSA 的架構下，網路端的功能可以藉由定義不同的 Service Capability Features 來提供給應用程式使用。由應用程式所實作的服務，則可以透過標準的 OSA API 來接取 Service Capability。本論文設計與發展出以 CORBA 為基礎的 OSA 平台。我們說明如何將 OSA API 中所定義的介面與函式，透過 CORBA 中的 clients，stubs，servants，和 skeletons 來實作，以及在我們的 OSA 平台中如何設定 CORBA POA 和 ORB。最後，我們透過初始存取(initial access)的認證流程，來說明 CORBA 機制如何在我們的 OSA 實作中運作。

CCL OSA: A CORBA-based Open Service Access System

Student: Chia-Ming Chou Advisor: Prof. Yi-Bing Lin

Department of Computer Science and Information Engineering
National Chiao Tung University

Abstract

Open Service Access (OSA) is a flexible and efficient approach for mobile service creation and deployment. OSA allows network operators and enterprises to increase revenues via third party applications and service providers. In OSA, network functionality offered to applications is defined by a set of *Service Capability Features* (SCF). Services can be implemented by applications accessing the *Service Capability* (SC) through the standardized OSA *Application Programming Interface* (API). This thesis designs and develops a CORBA-based OSA system. We show how OSA API interfaces and functions can be implemented by CORBA clients, stubs, servants, and skeletons, and how CORBA POA and ORB are set up for our OSA implementation. Then we use the authentication procedure for initial access to illustrate how CORBA mechanism works for our OSA implementation.

Acknowledgements

I would like to express my sincere thanks to my advisor, Prof. Yi-Bing Lin. Without his supervision and perspicacious advice, I can not complete this thesis. I learned a lot from him. Thanks also to all colleagues in the Laboratory 117.

I also would like to express my thanks to all my friends who have accompanied me through happiness and tears and have created lots of wonderful experiences in my life.

Finally, I am grateful to my dear parents, my brother and my sister for their unfailing love and firmly support in these years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Existing telecommunications services are considered as a part of network operation's domain, and the development of services are achieved by, for example, *Intelligent Network* (IN) technology. By introducing Internet and mobility into the telecommunications networks, more flexible and efficient approaches are required for mobile service deployment. Such approaches must allow network operators and enterprises to increase revenues via third party applications and service providers. To achieve the above goals, standardization bodies such as 3GPP CN5, ETSI SPAN12, ITU-T SG11 and the Parlay Group have been defining *Open Service Access* (OSA) specifications [1]. OSA provides unified service creation and execution environments to speed up service deployment that is independent from the underlying mobile network technology. In OSA, network functionality offered to applications is defined by a set of *Service Capability Features* (SCF). Services can be implemented by applications accessing the *Service Capability* (SC) through the standardized OSA *Application Programming Interface* (API).

As illustrated in Figure 1.1, the OSA consists of three parts: *Applications* are implemented in one or more *Application Servers* (AS; Figure 1.1 (1)). *Framework* (FW; see Figure 1.1 (2)) authorizes applications to utilize the *Service Capabilities* (SC; Figure 1.1(5)) in the network. That is, an application can only access the OSA API via the FW for services. *Service Capability Servers* (SCS; Figure 1.1 (3)) provide the applications access to underlying network functionality through SCFs (Figure 1.1 (4)). These SCFs, specified in terms of interface classes and their methods, are offered by SCs within networks (and under network control). SCs are bearers needed to realize services.

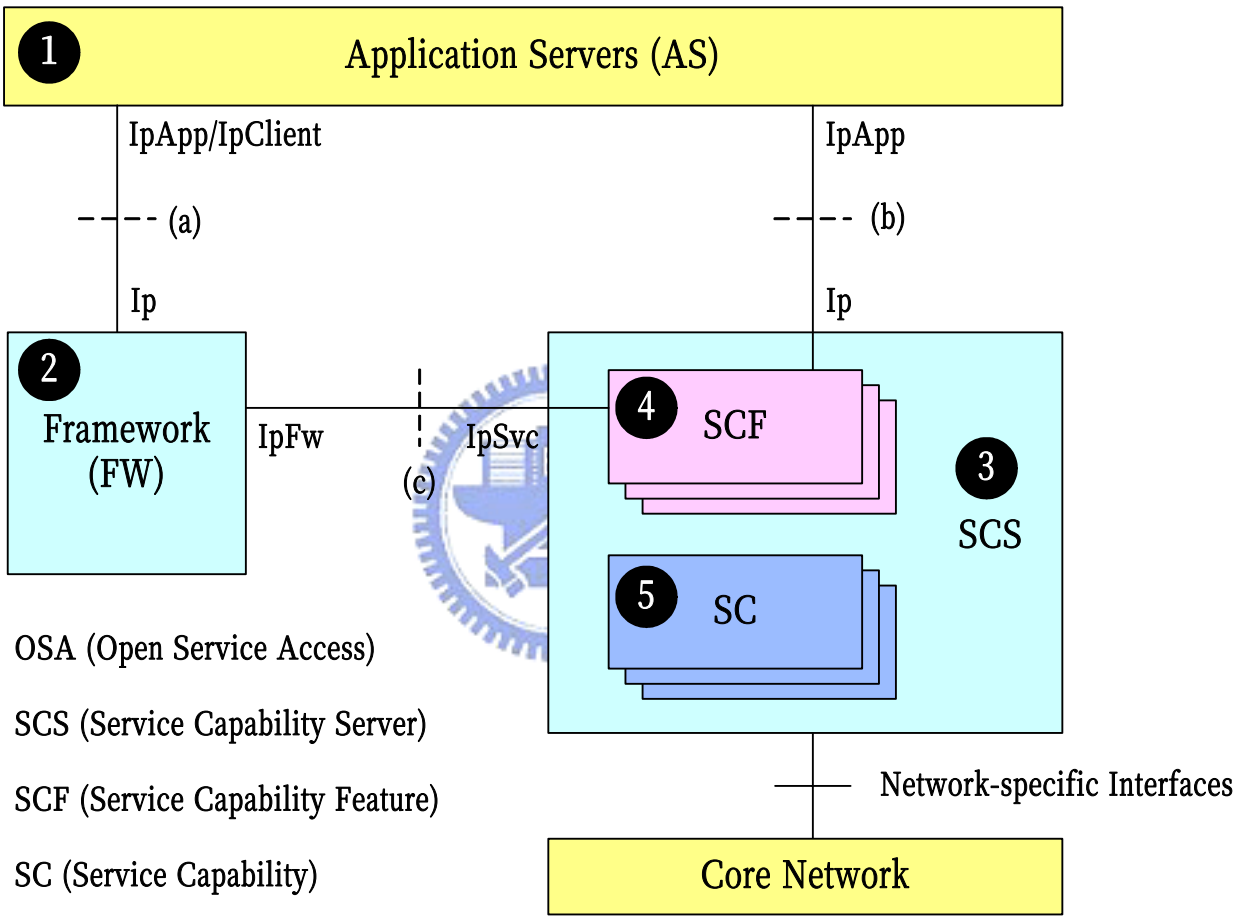


Figure 1.1 OSA Architecture

The FW is considered as one of the SCSs, and is always present, one per network. The FW provides access control functions to authorize the access to SCFs or service data for any API method invoked by an application, with specified security level, context, domain, etc. Before any application can interact with a network SCF, an off-line service agreement must be established. Once the service agreement exists, mutual authentication can be performed between the application and the FW. Then the application can be authorized by the FW to access a specific SCF (in OSA, authentication must precede authorization). Finally the application can use the *discovery* function to obtain information on authorized network SCFs. The discovery function can be used at any time after successful authentication. SCFs offered by an SCS are typically registered at the FW. This information is retrieved when the application invokes the discovery function. Framework allows OSA to go beyond traditional IN technology through openness, discovery, and integration of new features. Based on TINA [6], the FW provides controlled access to the API by supporting flexibility in application location and business scenarios. Furthermore, the FW allows multi-vendorship and even the inclusion of non-standardized APIs, which is crucial for innovation and service differentiation. An SCS can be deployed as a standalone node in the network or directly on a node in the core network. In the distributed approach, the OSA gateway node contains the FW and zero or more SCS components. Other SCSs are implemented in different nodes. It is possible to add more SCSs and distribute the load from different applications over multiple SCSs. At the service selection phase, the FW may divert one application to one SCS and another to a different SCS. With middleware such as CORBA, it is possible to distribute load on a session basis without the application being aware that different sessions involve different SCSs. In

some APIs, it is possible to add multiple application callbacks to the SCS so that the SCS can distribute the load of multiple sessions over different applications running on different servers. To allow applications from visited networks to use the SCSs in the home network, all communications between the application server and the SCSs must be secured through, e.g., *Secure Socket Layer* or IPSec.

Examples of OSA SCFs are given as follows: *Call and session control SCFs* provide capabilities for setting up basic calls or data sessions as well as manipulating multimedia conference calls. *User and terminal related SCFs* allow obtaining information from the end-user (including user location and status) and the terminal capabilities, playing announcements, sending short text messages, accessing to mailboxes, and so on. *Management related SCFs* provision connectivity QoS, access to end-user account and application/data usage charging. Interaction between an application and an SCS is always initiated by the application. In some scenarios, it is required to initiate the interaction from the SCS. An example is the *call screening* service. Suppose that the network routes a call to a user who has subscribed to this OSA service. Before the call reaches the user, the call screening application needs to be invoked. This issue is resolved by the OSA *request of event notification* mechanism. Initially, the application issues an OSA interface class method (API call) to the SCS. This OSA method allows the SCS to invoke the application (e.g., call screening) through a callback function when it receives events (e.g., incoming calls) from the network related to the application.

Since functionality inside a telecommunications network can be accessible via the OSA APIs, applications can access different network capabilities using a uniform programming paradigm. To be accessible to a side developer community, the APIs should be deployed based on open *information technology* (IT). The details will be elaborated in the next chapter. Three OSA API classes are defined among the applications, the FW, and the SCFs (in the SCSs).

1. Interface classes between the applications and the FW (Figure 1.1(a)) provide control of access to the network and integrity management; specifically, they provide applications with functions such as authentication, authorization, and discovery of network functionality. The FW-side interfaces to be invoked by the applications are prefixed with "IpFw". The application-side interfaces to be called back by the FW are prefixed with "IpApp" or "IpClient".

2. Interface classes between the applications and the SCFs (Figure 1.1 (b)) allow the applications to invoke network functionality for services. The SCF-side interfaces to be invoked by the applications are prefixed with "IpSvc". The application-side interfaces to be called back by the SCFs are prefixed with "IpApp".

3. Interface classes between the FW and the SCFs (Figure 1.1 (c)) provide the mechanisms for SCF registration and a multi-vendor environment. The SCF-side interfaces to be used by the FW are prefixed with "IpSvc". The FW-side interfaces to be used by the SCFs are prefixed with "IpFw".

The SCSs implement the OSA server side of the API and the applications implement the OSA client side of the API. An application should communicate with an SCS through standard IT middleware infrastructure such as *Common Object Request Broker Architecture* (CORBA) [7]. In a research collaboration between National Chiao Tung University and Computer and Communications Laboratories (CCL)/Industrial Technology Research Institute (ITRI), we have developed the CCL OSA system. Details of the CORBA-based APIs implemented in CCL OSA are given in the next chapter.

This thesis is organized as follows. In Chapter 2, we show how OSA API interfaces and functions can be implemented by CORBA clients, stubs, servants, and skeletons, and how CORBA POA and ORB are set up for our OSA implementation. In Chapter 3, we use the authentication procedure for initial access to illustrate how the Application Server and the Framework authenticate with each other. In Chapter 4, we take the authentication procedure for initial access as example to show how CORBA mechanism works for our OSA implementation.

Chapter 2

CORBA-based OSA API

CORBA is an emerging open distributed object computing infrastructure, which provides the higher layers a uniform view of underlying heterogeneous network and OS layers. CORBA automates many common network programming tasks such as object registration, location, activation, request demultiplexing, framing and error-handling. Besides, programming language and operation system independence provides a solid basis for both the integration of legacy systems and the development of new applications.

In this chapter, we introduce the CORBA Architecture and how OSA API interfaces and functions can be implemented by CORBA clients, stubs, servants, and skeletons. Then we demonstrate the functionality of each POA policy and show how to apply them in our implementation. Finally, we discuss different POA approaches in terms of space-time trade-offs.

2.1 Introduction to CORBA Architecture

Figure 2.1 illustrates the CORBA architecture.

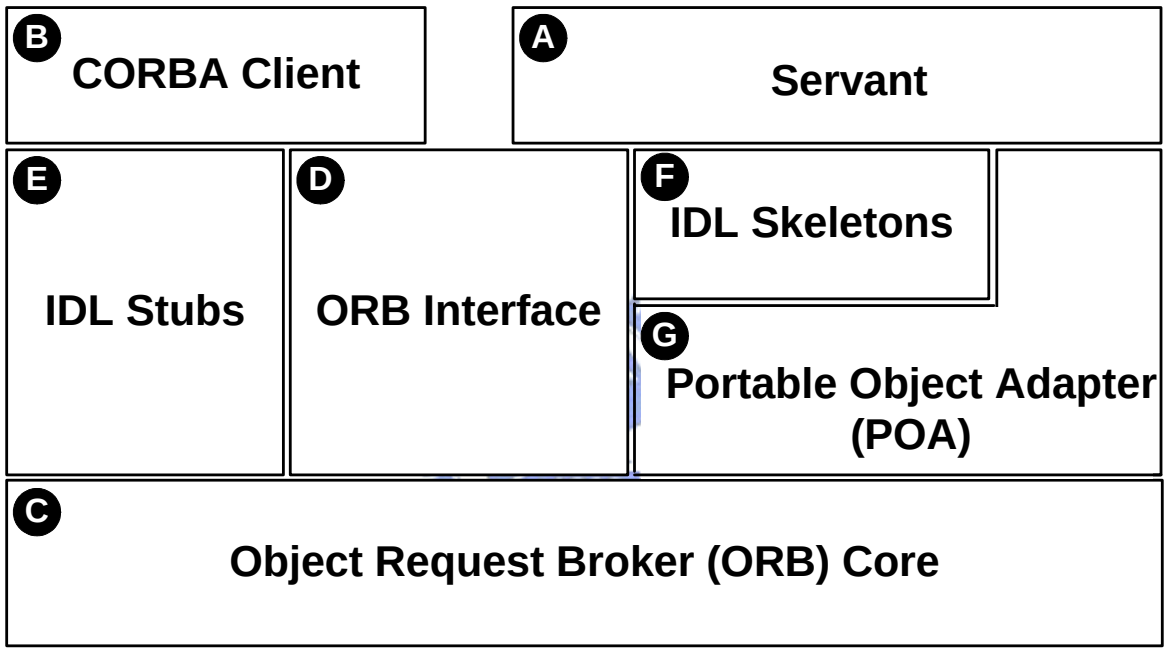


Figure 2.1 CORBA Architecture

In this architecture an *object* is a CORBA programming entity that consists of an identity, an interface, and an implementation known as *servant* (Figure 2.1 (A)). An object reference uniquely identifies the object across servers. This reference associates the object with one or more servant implementations. In CCL OSA API, every CORBA object is associated with one servant.

A *Servant* is an implementation programming language entity that defines the operations to support an *Object Management Group* (OMG) *Interface Definition Language* (IDL) interface. Servants can be written in languages such as C, C++ or Java.

A *Client* (Figure 2.1 (B)) is a program entity that invokes an operation on a servant. The client performs application tasks by invoking operations on object references. An object can be remote or local to the client. In CORBA, accessing a remote object should be as simple as calling an operation on a local object. A client always interacts with a servant through the corresponding object reference. Note that the CORBA concepts of client and servant are different from that of applications and servers in OSA (to be elaborated in Chapters 3 and 4).

Object Request Broker (ORB) is a logical entity that can be implemented through several alternatives (such as one or more processes or a set of libraries). In CCL OSA, ORB is implemented as libraries. An ORB consists of an *ORB Core* (Figure 2.1 (C)) and an *ORB interface* (Figure 2.1 (D)). The ORB simplifies distributed programming by decoupling the client from the details of the method invocation. This makes client requests appear to be local

procedure calls. The ORB Core provides a mechanism for transparent delivery requests from clients to target servants. When a client invokes an operation, the ORB Core is responsible for finding the servant, transparently activating it if necessary, delivering the request to the object, and returning any response to the client. An ORB Core is typically implemented as a run-time library linked into both client and server applications.

To decouple applications from implementation details, the CORBA specification defines an abstract ORB interface that provides various helper functions such as converting object references to strings. Specifically, CORBA *Interface Definition Language* (IDL) *stubs* (Figure 2.1 (E)) and *skeletons* (Figure 2.1 (F)) glue the clients, servants, and the ORB. The CORBA IDL definitions are transformed into classes, structs, and functions in a particular language (e.g., C++, C, Java, etc). Such transformation is automated by a CORBA IDL compiler. In CCL OSA, the target language is JAVA. We note that an object is an instance of an IDL interface. The corresponding servants implement the operations defined by the IDL. Several interfaces have been defined by IDL stubs to provide a strongly-typed, *static invocation interface* (SII) that marshals application (client) parameters into a common data-level representation. On the other hand, *skeletons* demarshal the data-level representation back into typed parameters that are meaningful to an application (server).

Portable Object Adapter (POA; Figure 2.1 (G)) is a CORBA portability enhancement. POA enables ORBs to support various types of servants that process similar requirements [7]. POA associates an IDL servant with objects, demultiplexes incoming requests to the servant, and dispatches the appropriate operation on that servant. The POA design results in a smaller and simpler ORB that can still support a wide range of object granularities, lifetimes, policies, implementation styles, and so on.

2.2 The POA Policies

Several POA policies are specified in CCL OSA. *Threading policy* specifies the POA threading model. A POA can either be single-threaded or multi-threaded concurrently controlled by the ORB. CCL OSA implementation uses ORB controlled multi-threading model.

Lifespan policy specifies whether the CORBA objects created within a POA are *persistent* or *transient*. A transient object is destroyed when the process creating the object terminates. On the other hand, a persistent object can live beyond the life time of the process that created it. All POAs are transient. In any CORBA system, there is a root POA called rootpoa. In CCL OSA, rootpoa creates several POAs with the persistent life span policy. An example is POA ipinitialpoa created for the FW. In CCL OSA, rootpoa creates transient objects. For example, in Initial Access service, an authentication object (with the reference ipAPIlevelauthentication_ref) is transient because this object is session-oriented and is destroyed when the authentication procedure is complete. On the other hand, the FWInitialContact object (with the reference ipinitial_ref; see Step 1.1 in Chapter 3) created by ipinitialpoa is persistent and should not be destroyed after the authentication action (between an AS application and the FW) is complete. The FWInitialContact object is the first contact point for any applications to start the authentication procedure. Therefore it must be persistent and active at any time. When the CCL OSA recovers from a failure, all persistent objects will be re-activated.

Object Id uniqueness policy specifies whether the servants activated in a POA must have unique Object Ids. In CCL OSA, since every object is implemented with one servant, the servant always has a unique Object Id. *Object Id assignment policy* specifies whether the Object Ids in a POA are generated by the application or the ORB. In CCL OSA, object Ids are generated by the ORB to avoid accidentally generating duplicated Object Ids by the programmer.

Implicit activation policy specifies whether implicit activation of servants is supported in a POA. With the implicit policy, the POA implicitly activates an object when the server application attempts to obtain a reference to a servant that is not already active.
In CCL OSA, objects in rootpoa are implicitly activated. These implicitly created transient objects can be destroyed at the end of the session of a client application, or are automatically cleaned up when the server process terminates. In CCL OSA POAs such as ipinitialpoa, objects are persistent, and should not be automatically clean up when the server process terminates. Therefore, these persistent objects are explicitly activated and should be explicitly destroyed by the programmer when he/she decides to stop providing the functionality.

Servant retention policy specifies whether the POA retains active servants in an active object map. A POA either retains the associations between servants and CORBA objects or establishes a new CORBA object/servant association for each incoming request. In CCL OSA, the POA retains the associations between servants and CORBA objects in an active object map.

Request processing policy specifies how requests are processed by the POA. The alternatives include (1) to consult its active object map only, (2) to use a default servant, or (3) to invoke a servant manager. There are two types of servant managers: A manager of the ServantActivator type follows the RETAIN policy. A manager of the ServantLocator type follows the NON_RETAIN policy. Since CCL OSA uses active object map, it uses first alternative for the request processing policy; i.e., we consult POA's active object map for request processing.

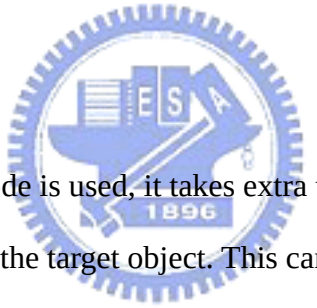
Table 2.1 lists the POA policy modes used in CCL OSA.

Table 2.1 : POA Policy Modes for CCL OSA

Policy	Mode of the policy
Thread Policy	ORB_CTRL_MODEL
Lifespan Policy	TRANSIENT (rootpoa) PERSISTENT (e.g. ipinitialpoa)
Object ID Uniqueness Policy	UNIQUE_ID
ID Assignment Policy	SYSTEM_ID
Servant Retention Policy	RETAIN
Request Processing Policy	USE_ACTIVE_OBJECT_MAP_ONLY
Implicit Activation Policy	IMPLICIT_ACTIVATION (for rootpoa) NO_ IMPLICIT_ACTIVATION (e.g. ipinitialpoa)

2.3 Space-Time Trade-Offs for Request Processing

The Servant retention policy and Request processing policy determine how the POA dispatches the request. From the description in the last section, we list five approaches based on the Servant retention policy modes (RETAIN and NON_RETAIN) and the Request processing policy modes (USE_ACTIVE_OBJECT_MAP_ONLY, USE_SERVANT_MANAGER and USE_DEFAULT_SERVANT). The approach with the NON_RETAIN mode and the USE_ACTIVE_OBJECT_MAP_ONLY mode does not exist because the POA with the NON_RETAIN mode does not retain active servants in its active object map.



We note that if the RETAIN mode is used, it takes extra time for the POA to locate a servant associated with the ObjectId of the target object. This can be achieved by looking up the associated entries (servant and its corresponding ObjectId) stored in the active object map. The space complexity depends on the size of the active object map.

On the other hand, if the NON_RETAIN mode is used, it takes extra time for the servant to determine which object it is incarnating for this request. No lookup in the active object map is required. Obviously, the space complexity for the NON_RETAIN mode is less than that for the RETAIN mode.

To simplify the analysis, we assume that the same request is invoked in each approach and the

time required to complete this request in each approach is also the same. We discuss the five approaches as follows:

Approach 1: RETAIN with USE_ACTIVE_OBJECT_MAP_ONLY

(1) For each incoming request, the POA only consults its active object map to find the corresponding servant. Therefore the time complexity depends on how the hash algorithm in the ORB is designed to find the servant.

(2) The space complexity includes the number of associations between servants and ObjectIds stored in the active object map of the POA. In other words, the number of entries stored in the active object map is the same as the number of objects hosted by the POA.

Approach 2: RETAIN with USE_SERVANT_MANAGER

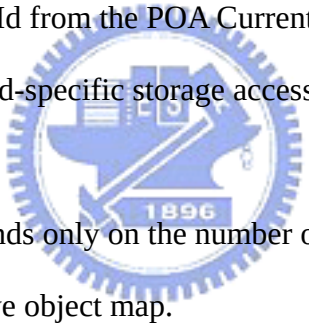
(1) For each incoming request, the POA first looks up its active object map to find the corresponding servant. If the POA can not find such an entry, it invokes its ServantActivator to obtain a servant. The obtained servant and its corresponding ObjectId is stored in an entry in the POA's active object map. Therefore the time complexity for this approach is higher than Approach 1.

(2) Like Approach 1, the space complexity of this approach includes the number of associations between servants and ObjectIds stored in the active object map of the POA. If not every object hosted by the POA has an opportunity to be invoked, we may not

necessary to store all associations in the active object map as required in Approach 1. Only in the worse case, all objects hosted by the POA are invoked. Therefore the number of entries in the active object map is the same as Approach 1.

Approach 3: RETAIN with USE_DEFAULT_SERVANT

(1) For each incoming request, the POA determines whether or not the entry (a pair of servant and its corresponding ObjectId) exists in the active object map. If the POA can not use the ObjectId obtained from the request to find the corresponding servant, the default servant is invoked to handle this request. In this approach, extra time is required for the default servant to obtain the ObjectId from the POA Current object (it is the context of the current running thread) due to thread-specific storage access.



(2) The space complexity depends only on the number of associations between servants and ObjectIds stored in the active object map.

Approach 4: NON_RETAIN with USE_SERVANT_MANAGER

(1) For each incoming request, the POA always invokes its ServantLocator to obtain a servant. The time complexity for obtaining a servant depends on how the ServantLocator is implemented.

(2) Because a POA with NON_RETAIN policy has no active object map, the space complexity is minimized. However, if this ServantLocator must create and destroy a new

servant on the heap for each request, this not only costs time but also increases the fragmentation of the heap. Using some sort of servant pool to manager servant instances provides better implementation of the ServantLocator.

Approach 5: NON_RETAIN with USE_DEFAULT_SERVANT

(1) For each incoming request, the POA always uses this default servant to process the request. The time complexity for finding this servant is minimized because the POA only needs to access its default servant. However, a default servant must always determine the target ObjectId from the POA Current object, so additional time is required to access thread-specific storage.

(2) The space complexity is minimized because the POA has no active object map.

Based on the above discussion, one should choose the appropriate policies according to the application. Approach 1 (the RETAIN and USE_ACTIVE_OBJECT_MAP_ONLY policies) used in our implementation allows simple and quick request processing at the cost of larger memory usage as compared with Approaches 2-5.

Chapter 3

OSA Mutual Authentication for Initial Access

As mentioned in Chapter 1, an OSA application must authenticate with the FW before it can access any SCFs. Figure 3.1 illustrates the message flow of the mutual authentication procedure for initial access in CCL OSA. As shown in Figure 4.1, each of the AS and the FW implements both CORBA clients and CORBA servants. In the OSA mutual authentication procedure, the AS CORBA Client uses the FW reference ipinitial_ref of Initial Contact interface IpInitial. In CCL OSA, this reference is obtained through a URL (e.g., corbaname::pcs.csie.nctu.edu.tw:3500#IpInitial) and the *naming service* of the FW. This *naming service* is a standard CORBA service that allows the CORBA client application to locate the object through a URL. IpInitial is implemented by the FW servant ipinitialimpl (Figure 3.1 (3)) to support the authentication function initiateAuthenticationWithVersion.

To authenticate the FW, the AS CORBA Client (Figure 3.1 (2)) invokes the challenge function in the FW interface IpAPILevelAuthentication (implemented by the servant ipAPIlevelauthenticationimpl; see Figure 3.1 (4)). To authenticate the AS application, the FW CORBA Client (Figure 3.1 (5)) invokes the challenge (callback) function in the AS interface IpClientAPILevelAuthentication (implemented by the servant ipclientAPIlevelauthenticationimpl; see Figure 3.1 (1)). The detailed steps are described as follows.

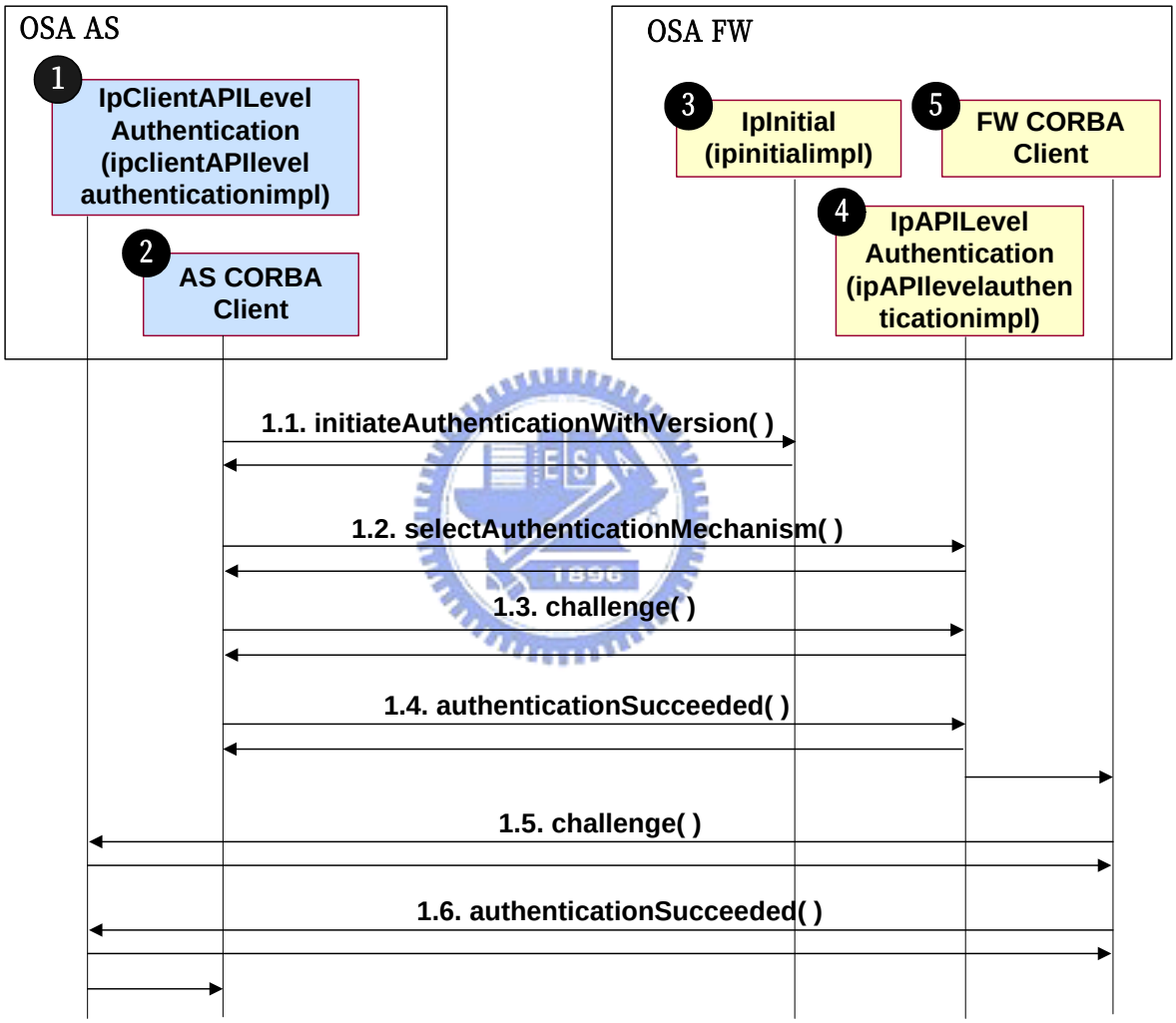


Figure 3.1 Message Flow for OSA Mutual Authentication

Step 1.1. The AS CORBA Client first generates the reference ipclientAPIlevelauthentication_ref of the AS interface IpClientAPILevelAuthentication. This reference will be called back by the FW to authenticate the AS application in Steps 1.5 and 1.6. The AS CORBA Client uses ipinitial_ref to invoke the FW function initiateAuthenticationWithVersion where the callback object reference ipclientAPIlevelauthentication_ref is included as a parameter. The FW servant ipinitialimpl returns a reference ipAPIlevelauthentication_ref of the FW interface IpAPILevelAuthentication. This FW interface is responsible for answering the authentication challenge from the OSA application.

Step 1.2. The AS CORBA Client selects the authentication algorithm by invoking the FW function selectAuthenticationMechanism of the reference ipAPIlevelauthentication_ref. In CCL OSA, authentication algorithm SHA-1 is utilized.

Step 1.3. The AS CORBA Client authenticates the FW by invoking the function challenge of the FW reference ipAPIlevelauthentication_ref. The challenge function takes a random number as the parameter (in the byte-stream format). The FW servant ipAPIlevelauthenticationimpl executes the SHA-1 algorithm using the received random number and returns the result fw_digest to the AS CORBA Client.

Step 1.4. The AS CORBA Client validates the result fw_digest. Suppose that the authentication is successful, the AS CORBA Client invokes the function authenticationSucceeded of ipAPIlevelauthentication_ref to inform the FW servant ipAPIlevelauthenticationimpl of the result. Then the servant ipAPIlevelauthenticationimpl activates the FW CORBA Client to authenticate the AS application.

Step 1.5. The FW CORBA Client invokes the function challenge in the AS (callback) reference ipclientAPIlevelauthentication_ref (obtained in Step 1.1). Similar to Step 1.3, this function takes a random number as the parameter. The AS servant ipclientAPIlevelauthenticationimpl executes the SHA-1 algorithm and returns the result app_digest to the FW CORBA Client.

Step 1.6. The FW CORBA Client validates the result app_digest. Suppose that the authentication is successful, the FW CORBA Client invokes the function authenticationSucceeded of ipclientAPIlevelauthentication_ref to inform the AS servant ipclientAPIlevelauthenticationimpl of the result. This result is passed to the AS CORBA Client.

At this point, the mutual authentication procedure is complete, and the OSA CORBA Client can access SCFs through the FW.

Chapter 4

Detailed CORBA Interactions for CCL OSA

We use the initiateAuthenticationWithVersion function invoked from the AS to the FW (see Step 1.1 in Figure 3.1) and the callback challenge function invoked from the FW to the AS (see Step 1.5 in Figure 3.1) as examples to illustrate the CORBA interaction between the AS and the FW.

As we mentioned in the previous chapter, before the AS CORBA Client requests mutual authentication, it first retrieves the FW reference ipinitial_ref of the IpInitial interface. The CORBA behavior for invoking initiateAuthenticationWithVersion is described in Steps 2.1-2.9 (which implement Step 1.1 in the previous chapter).

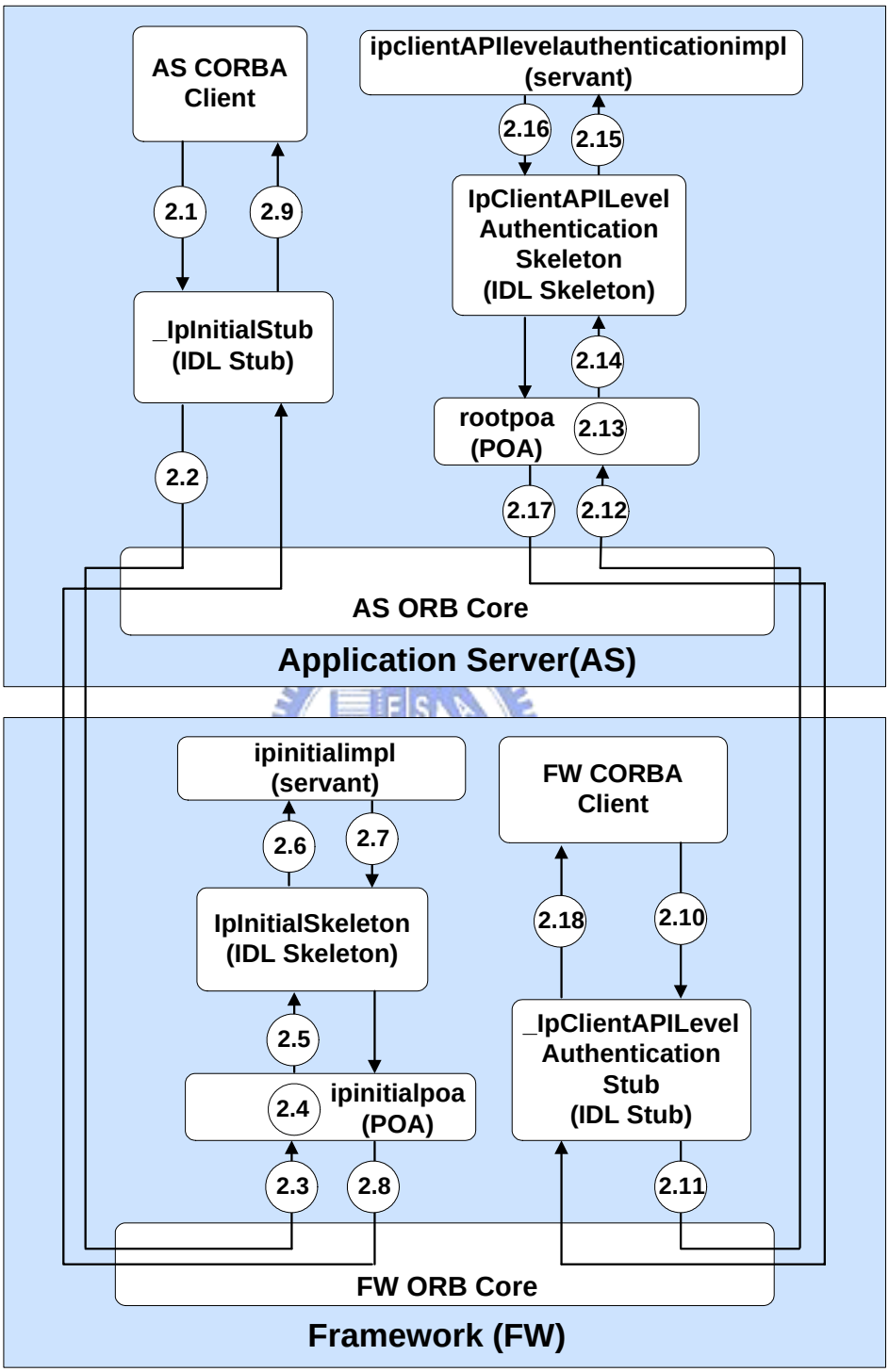
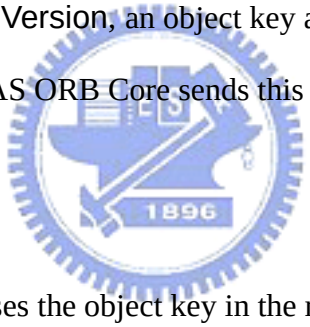


Figure 4.1 CORBA Interaction Flows between the AS and the FW

Step 2.1. The AS CORBA Client uses ipinitial_ref to invoke the FW function initiateAuthenticationWithVersion where the callback object reference ipclientAPIlevelauthentication_ref is included as a parameter. This request is sent to the AS stub _IpInitialStub.

Step 2.2. The stub _IpInitialStub marshals the parameters clientDomain, authType, and frameworkVersion into the *common data representation* (CDR) format and forwards the CDR data and the operation name initiateAuthenticationWithVersion to the AS ORB Core. The AS ORB Core uses the CDR data, the operation name initiateAuthenticationWithVersion, an object key and other information to construct a request message. Then the AS ORB Core sends this request message to the FW ORB Core.



Step 2.3. The FW ORB Core uses the object key in the request message to locate the target POA ipinitialpoa and delivers the request message to ipinitialpoa.

Steps 2.4. and 2.5. The FW POA ipinitialpoa uses the received object key to locate the servant ipinitialimpl in the ipinitialpoa's active object map, and then activates the skeleton IpInitialSkeleton.

Step 2.6. The skeleton demarshals the parameters in the request message into arguments. Three arguments clientDomain (containing the callback reference

ipclientAPIlevelauthentication_ref to be used in Step 2.10), authType (P_OSA_AUTHENTICATION) and frameworkVersion (FWv1.0) are passed as parameters to the function initiateAuthenticationWithVersion of the servant ipinitialimpl.

Step 2.7. The servant ipinitialimpl performs the function initiateAuthenticationWithVersion and returns the result or exceptions to the skeleton IpInitialSkeleton.

Step 2.8. IpInitialSkeleton marshals the related results returned by the servant into the CDR format and forwards the CDR data to the FW ORB Core. The FW ORB Core uses the CDR data and other information to construct a reply message, and sends the reply message back to the AS ORB Core.

Step 2.9. The stub _IpInitialStub demarshals the results in the reply message, and returns the result (i.e, ipAPIlevelauthentication_ref; see Step 1.1) to the AS CORBA Client.

The CORBA behavior for invoking the callback challenge from the FW to the AS is described in Steps 2.10-2.18 (which implement Step 1.5 in the previous chapter).

Step 2.10. The FW CORBA Client uses ipclientAPIlevelauthentication_ref (obtained from the argument clientDomain in Step 2.6) to invoke the callback function challenge. This request is sent to the FW stub _IpClientAPILevelAuthenticationStub.

Step 2.11. The stub _IpClientAPILevelAuthenticationStub marshals the parameter challenge into the CDR format, and forwards the CDR data and the operation name challenge to the FW ORB Core. The FW ORB Core uses the CDR data, the operation name challenge, an object key and other information to construct a request message. Then the FW ORB Core sends this request message to the AS ORB Core.

Step 2.12. The AS ORB Core uses the object key in the request message to locate and delivers the request message to the target POA rootpoa.

Steps 2.13. and 2.14. POA rootpoa uses the received object key to locate the servant ipclientAPIlevelauthenticationimpl in its active object map, and activates the skeleton IpClientAPILevelAuthenticationSkeleton.

Step 2.15. Upon receipt of the request message, the AS skeleton IpClientAPILevelAuthenticationSkeleton retrieves the argument challenge and instructs the servant ipclientAPIlevelauthenticationimpl to execute the function challenge.

Step 2.16. After the function challenge is executed, the AS servant ipclientAPIlevelauthenticationimpl returns the result to the skeleton IpClientAPILevelAuthenticationSkeleton.

Step 2.17. IpClientAPILevelAuthenticationSkeleton marshals the related results returned by the servant into the CDR format and forwards the CDR data to the AS ORB Core. The AS ORB Core uses the CDR data to construct a reply message, and sends the reply message back to the FW ORB Core.

Step 2.18. The FW stub _IpClientAPILevelAuthenticationStub demarshals the result (app_digest) in the reply message and returns the result to the FW CORBA Client.

The above two examples illustrate how CCL OSA interaction can be implemented using CORBA technology.

Chapter 5

Conclusion

This thesis described a CORBA-based OSA system we designed and developed in CCL/ITRI. We showed how OSA API interfaces and functions can be implemented by CORBA clients, stubs, servants, and skeletons. We described how CORBA POA and ORB are set up for CCL OSA. Then we used the authentication procedure for initial access to illustrate how CORBA mechanism works for CCL OSA. One of the challenges for future OSA extension is to deploy open API-based services that support both legacy circuit-switched networks and all-IP based networks.

Bibliography

[1] Moerdijk, A.-J., and Klostermann, L. Opening the Networks with Parlay/OSA: Standards and Aspects Behind the APIs, IEEE Network, May/June, 2003.

[2] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Virtual Home Environment/Open Service Access; 3GPP TS 23.127 V6.0.0, 2002

[3] 3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API); Part 1: Overview; 3GPP TS 29.198-1 V6.0.1, 2004

[4] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Service Requirement for the Open Services Access (OSA); Stage 1; 3GPP TS 22.127 V6.4.0, 2004

[5] Walkden, M., Edwards, N., Foster, D., Jankovic, M., Odadzic, B., Nygreen, G., Moiso, C., Tognon, S.M., de Bruijn, G. Open Service Access: Advantages and opportunities in service provisioning on 3G Mobile Networks Definition and Solution of proposed Parlay/OSA Specification issues, Project P1110 Technical Information EDIN 0266-1110, 2002.

[6] Berndt, H., et al. TINA: Its Achievements and Its Future Directions, IEEE Communications Surveys and Tutorials, 3(1), 2000.

[7] Pyarali, I., and Schmidt, D.C. An Overview of the CORBA Portable Object Adapter, ACM StandardView Magazine, Volume 6, Issue1 (March,1998)

[8] Schmidt, D.C. and Vinoski, S. Object Adapter: Concepts and Terminology, C++ Report,

SIGS, Vol. 9, No 11 (Octorber,1997)

[9] Schmidt, D.C. and Vinoski, S. Using the Portable Object Adapter for Transient and Persistent CORBA Objects, C++ Report, SIGS, Vol. 10, No 4 (April, 1998)

[10] Schmidt, D.C. and Vinoski, S. C++ Servant Managers for the Portable Object Adapter, C++ Report, SIGS, Vol. 10, No 8 (September,1998)

[11] Schmidt, D.C. and Vinoski, S. C++ Servant Classes for the POA, C++ Report, SIGS, Vo1. 10, No 6 (June,1998)

[12] Bos, L. and Leroy, S. Toward an All-IP-Based UMTS System Architecture, IEEE Network, Jan. 2001, pp.36-45

[13] Depaoli, R. and Moiso, C. Network Intelligence for GPRS, IEEE Intelligent Network Workshop 2001

[14] Laitinen, M. and Rantala, J. Integration of Intelligent Network Services into Future GSM Networks, IEEE Communications, June. 1995

[15] Jabbari, B. Intelligent Network Concepts in Mobile Communications, IEEE Communications, Feb. 1992

[16] Pailer, R. and Stadker, J. A Service Framework for Carrier Grade Multimedia Services using PARPLAY APIs over a SIP System, Proceedings of the first workshop on Wireless mobile internet, July 2001

[17] Pailer, R., Stadker, J. and Miladinovic, I. Using PARLAY APIs Over a SIP System in a Distributed Service Platform for Carrier Grade Multimedia Services, Wireless Networks, Vol. 9, Issue 4, p.353-363, July 2003

[18] Maarten Wegdam, Dirk-Jaap Plas, Musa Unmehopa. Validation of the Open Service

Access API for UMTS Application Provisioning. In Proceedings of the 6th International Conference on Protocols for Multimedia Systems (PROMS 2001), Springer Verlag, October 17-19, 2001, Enschede, The Netherlands.

[19] Zeiss, J. Cross-Domain Service Deployment with OSA/Parlay and CORBA Components, KiVS2003 Conference, VDE Verlag, Feb. 2003

[20] Gross, J., Wegscheider, F. and Zeiss, J. Corba Component Based Implementation of Telecom Service Building Blocks, 7-th International Enterprise Distributed Object Computing Conference (EDOC), September 2003

Appendix A

Source Codes of Mutual Authentication Procedure

Table A.1 Source Program files of the FW side in the CCL OSA

Module	File Name	Function Description
FW Initial Module	IpInitialSkeleton.java	Skeleton file of the class IpInitial
	IpInitialImpl.java	Servant file of the class IpInitial
	IpInitialServer.java	Persistent Server for Initial Access Service
FW Authentication Module	IpAPILevelAuthenticationSkeleton.java	Skeleton file of the class IpAPILevelAuthentication
	IpAPILevelAuthenticationImpl.java	Servant file of the class IpAPILevelAuthentication
	FWClient.java	FW Authentication Client
	_IpClientAPILevelAuthenticationStub.java	Stub file of the class IpClientAPILevelAuthentication

Table A.2 Source Program files of the AS side in the CCL OSA

Module	File Name	Function Description
AS Initial Module	_IpInitialStub.java	Stub file of the class IpInitial
AS Authentication Module	IpClientAPILevelAuthenticationSkeleton.java	Skeleton file of the class IpClientAPILevelAuthentication
	IpClientAPILevelAuthenticationImpl.java	Servant file of the class IpClientAPILevelAuthentication
	ASClient.java	AS Authentication Client
	_IpAPILevelAuthenticationStub.java	Stub file of the class IpAPILevelAuthentication

A.1 FW Initial Module

```
/**
* Name: IpInitialSkeleton.java
* org/csapi/fw/fw_access/trust_and_security/IpInitialSkeleton.java .
* Generated by the IDL-to-Java compiler (portable), version "3.1"
* idlj –fall –skeletonName %Skeleton fw_if_access.idl
* from fw_if_access.idl
*/

package org.csapi.fw.fw_access.trust_and_security;
public abstract class IpInitialSkeleton extends org.omg.PortableServer.Servant implements
      org.csapi.fw.fw_access.trust_and_security.IpInitialOperations, org.omg.CORBA.portable.InvokeHandler
{
       // Constructors
       private static java.util.Hashtable _methods = new java.util.Hashtable ();
       static
       {
              _methods.put ("initiateAuthentication", new java.lang.Integer (0));
              _methods.put ("initiateAuthenticationWithVersion", new java.lang.Integer (1));
       }
       public org.omg.CORBA.portable.OutputStream _invoke (String $method,
              org.omg.CORBA.portable.InputStream in, org.omg.CORBA.portable.ResponseHandler $rh)
       {
              org.omg.CORBA.portable.OutputStream out = null;
```

```
java.lang.Integer __method = (java.lang.Integer)_methods.get ($method);
if (__method == null)
	throw new org.omg.CORBA.BAD_OPERATION (
	0,org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);
switch (__method.intValue ())
{
	case 0:
	/*org/csapi/fw/fw_access/trust_and_security/IpInitial/initiateAuthentication*/
	{
		try {
			org.csapi.fw.TpAuthDomain clientDomain =
				org.csapi.fw.TpAuthDomainHelper.read (in);
			String authType = org.csapi.fw.TpAuthTypeHelper.read (in);
			org.csapi.fw.TpAuthDomain $result = null;
			$result = this.initiateAuthentication (clientDomain, authType);
			out = $rh.createReply();
			org.csapi.fw.TpAuthDomainHelper.write (out, $result);
		} catch (org.csapi.TpCommonExceptions $ex) {
			out = $rh.createExceptionReply ();
			org.csapi.TpCommonExceptionsHelper.write (out, $ex);
		} catch (org.csapi.fw.P_INVALID_DOMAIN_ID $ex) {
			out = $rh.createExceptionReply ();
			org.csapi.fw.P_INVALID_DOMAIN_IDHelper.write (out, $ex);
		} catch (org.csapi.P_INVALID_INTERFACE_TYPE $ex) {
			out = $rh.createExceptionReply ();
			org.csapi.P_INVALID_INTERFACE_TYPEHelper.write (out, $ex);
		} catch (org.csapi.fw.P_INVALID_AUTH_TYPE $ex) {
			out = $rh.createExceptionReply ();
			org.csapi.fw.P_INVALID_AUTH_TYPEHelper.write (out, $ex);
		}
		break;
	}//end of case 0

	case 1:
	/*org/csapi/fw/fw_access/trust_and_security/IpInitial/initiateAuthenticationWithVersion*/
	{
		try {
			org.csapi.fw.TpAuthDomain clientDomain =
				org.csapi.fw.TpAuthDomainHelper.read (in);
			String authType = org.csapi.fw.TpAuthTypeHelper.read (in);
			String frameworkVersion = org.csapi.TpVersionHelper.read (in);
			org.csapi.fw.TpAuthDomain $result = null;
			$result = this.initiateAuthenticationWithVersion (clientDomain, authType,
				frameworkVersion);
			out = $rh.createReply();
			org.csapi.fw.TpAuthDomainHelper.write (out, $result);
		} catch (org.csapi.TpCommonExceptions $ex) {
			out = $rh.createExceptionReply ();
			org.csapi.TpCommonExceptionsHelper.write (out, $ex);
		} catch (org.csapi.fw.P_INVALID_DOMAIN_ID $ex) {
```

```
                        out = $rh.createExceptionReply ();
                        org.csapi.fw.P_INVALID_DOMAIN_IDHelper.write (out, $ex);
                    } catch (org.csapi.P_INVALID_INTERFACE_TYPE $ex) {
                        out = $rh.createExceptionReply ();
                        org.csapi.P_INVALID_INTERFACE_TYPEHelper.write (out, $ex);
                    } catch (org.csapi.fw.P_INVALID_AUTH_TYPE $ex) {
                        out = $rh.createExceptionReply ();
                        org.csapi.fw.P_INVALID_AUTH_TYPEHelper.write (out, $ex);
                    } catch (org.csapi.P_INVALID_VERSION $ex) {
                        out = $rh.createExceptionReply ();
                        org.csapi.P_INVALID_VERSIONHelper.write (out, $ex);
                    }
                    break;
                }// end of case 1

                default:
                    throw new org.omg.CORBA.BAD_OPERATION (0,
                    org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);
            }// end of switch
                return out;
        } // _invoke

        // Type-specific CORBA::Object operations
        private static String[] __ids = {
            "IDL:org/csapi/fw/fw_access/trust_and_security/IpInitial:1.0",
            "IDL:org/csapi/IpInterface:1.0"
        };
        public String[] _all_interfaces (org.omg.PortableServer.POA poa, byte[] objectId)
        {
            return (String[])__ids.clone ();
        }
        public IpInitial _this()
        {
            return IpInitialHelper.narrow(super._this_object());
        }
        public IpInitial _this(org.omg.CORBA.ORB orb)
        {
            return IpInitialHelper.narrow(
            super._this_object(orb));
        }
} // class IpInitialSkeleton

/**
* Name: IpInitialImpl.java
* This is the servant file which implementes the function defined in the idl file.
*/

import org.csapi.*;
import org.csapi.fw.*;
import org.csapi.fw.fw_access.trust_and_security.*;
```

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
package com.fwimpl;
public class IpInitialImpl extends org.csapi.fw.fw_access.trust_and_security.IpInitialSkeleton
{
        private ORB orb;
        private InitConfig tool;
        pricate String prefix=null;//" format:    corbaname::host_address:port_number#";
        public IpInitialImpl(ORB orb)
        {
                this.orb=orb;
                tool = new FwTool();
                this.prefix = tool.getPrefix();
        }
        public org.csapi.fw.TpAuthDomain initiateAuthentication (org.csapi.fw.TpAuthDomain clientDomain,
                String authType)
        {
                System.out.print("Received the initiateAuthentication request from "+
                                clientDomain.DomainID.ClientAppID() + " with Authentication type ");
                System.out.print("This method is already deprecated!");
                return null;
        }

        public org.csapi.fw.TpAuthDomain initiateAuthenticationWithVersion
                (org.csapi.fw.TpAuthDomain clientDomain, String authType, String frameworkVersion)
        {

                System.out.print("Received the IpInitial request from "+ clientDomain.DomainID.ClientAppID() + "
                with Authentication type ");
                System.out.println(authType);
                //prepairing the Authentication Interface
                try{
                        org.omg.CORBA.Object obj = orb.string_to_object(prefix+"FwObjectFactory");
                        FwObjectFactory FwObjectFactoryimpl = FwObjectFactoryHelper.narrow(obj);
                        //get the callback reference clientDomain.AuthInterface from the parameter passed through the
                        // function
                        IpAPILevelAuthentication ipAPIlevelauthenticationimpl =
                                new IpAPILevelAuthenticationImpl(orb, clientDomain.AuthInterface);
                        System.out.println("Obtained a IpInterface handle on server object ");
                        TpDomainID fwDomainID = new TpDomainID();
                        fwDomainID.FwID("CCL_OSA_Framework");
                        IpInterface fwInterface = ipAPIlevelauthenticationimpl;
                        TpAuthDomain fwDomain = new TpAuthDomain(fwDomainID, fwInterface);
                        return fwDomain;
                }catch(Exception e){
                        e.printStackTrack(); //logging the error message!
                }
```

```
            return null;
        }
}// class IpInitialImpl

/**
* Name: IpInitialServer.java
* This is the persistent server for IpInitial Access Service.
*/

import org.csapi.*;
import org.csapi.fw.*;
import org.csapi.fw.fw_access.trust_and_security.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.*;
public class IpInitialServer
{
        public static void main(String args[] )
        {
                InitConfig tool = new InitConfig();
                String host=tool.getOrbdHost();
                String port=tool.getOrbdPort();
                System.out.println("orb_host="+host);
                System.out.println("orb_port="+port);
                Properties properties = System.getProperties();
                properties.put( "org.omg.CORBA.ORBInitialHost",host);
                 properties.put( "org.omg.CORBA.ORBInitialPort",port);
                try {
                        //initialize the ORB
                        ORB orb = ORB.init(args, properties);
                        IpInitialImpl ipinitialimpl = new IpInitialImpl(orb);
                        //locate the rootpoa in the CORBA System
                        POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
                        //Create the Persistent Policy
                        Policy[] policy = new Policy[1];
                        policy[0] = rootpoa.create_lifespan_policy(LifespanPolicyValue.PERSISTENT);
                        //Create a persistent POA “ipinitialpoa” by passing the policy
                        POA ipinitialpoa = rootpoa.create_POA("ipinitialpoa ", null, policy );
                        /* Activate PersistentPOA's POAManager
                        * Without this all calls to persistent server will hang,
                        * because POAManager will be in the 'HOLD' state
                        */
                        ipinitialpoa.the_POAManager().activate( );
                        //Associate the servant with PersistentPOA
                        ipinitialpoa.activate_object( ipinitialimpl );
                        // Resolve RootNaming context and bind a name for the servant
                        org.omg.CORBA.Object nsobj = orb.resolve_initial_references("NameService" );
```

```
                NamingContextExt rootContext = NamingContextExtHelper.narrow(obj);
                //bind the object reference in the naming context
                NameComponent[] nc = rootContext.to_name("IpInitial");
                rootContext.rebind(nc, ipinitialpoa.servant_to_reference(ipinitialimpl));
                //wait for client requests
                orb.run();
                System.out.println("IpInitialServer is running");
            }catch (Exception e) {
                System.err.println("Exception in IpInitialServer Startup " + e);
            }
        }// end of main()
}
```

A.2 FW Authentication Module

```
/**
* Name: IpAPILevelAuthenticationSkeleton.java
* org/csapi/fw/fw_access/trust_and_security/IpAPILevelAuthenticationSkeleton.java .
* Generated by the IDL-to-Java compiler (portable), version "3.1"
* idlj –fall –skeletonName %Skeleton fw_if_access.idl
* from fw_if_access.idl
*/

package org.csapi.fw.fw_access.trust_and_security;
public abstract class IpAPILevelAuthenticationSkeleton extends org.omg.PortableServer.Servant implements
        org.csapi.fw.fw_access.trust_and_security.IpAPILevelAuthenticationOperations, o
        rg.omg.CORBA.portable.InvokeHandler
{
        private static java.util.Hashtable _methods = new java.util.Hashtable ();
        //static initialization block
        static
        {
                _methods.put ("selectEncryptionMethod", new java.lang.Integer (0));
                _methods.put ("authenticate", new java.lang.Integer (1));
                _methods.put ("abortAuthentication", new java.lang.Integer (2));
                _methods.put ("authenticationSucceeded", new java.lang.Integer (3));
                _methods.put ("selectAuthenticationMechanism", new java.lang.Integer (4));
                _methods.put ("challenge", new java.lang.Integer (5));
                _methods.put ("requestAccess", new java.lang.Integer (6));
        }
        public org.omg.CORBA.portable.OutputStream _invoke (String $method,
                org.omg.CORBA.portable.InputStream in, org.omg.CORBA.portable.ResponseHandler $rh)
        {
                org.omg.CORBA.portable.OutputStream out = null;
                java.lang.Integer __method = (java.lang.Integer)_methods.get ($method);
                if (__method == null)
                        throw new org.omg.CORBA.BAD_OPERATION
                        (0, org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);
                switch (__method.intValue ())
                {
```

```
case0:
/*org/csapi/fw/fw_access/trust_and_security/IpAPILevelAuthentication/
*selectEncryptionMethod
*/
{
        try
        {
                String encryptionCaps = org.csapi.fw.TpEncryptionCapabilityListHelper.read (in);
                String $result = null;
                $result = this.selectEncryptionMethod (encryptionCaps);
                out = $rh.createReply();
                out.write_string ($result);
        } catch (org.csapi.TpCommonExceptions $ex) {
                out = $rh.createExceptionReply ();
                org.csapi.TpCommonExceptionsHelper.write (out, $ex);
        } catch (org.csapi.fw.P_ACCESS_DENIED $ex) {
                out = $rh.createExceptionReply ();
                org.csapi.fw.P_ACCESS_DENIEDHelper.write (out, $ex);
        } catch (org.csapi.fw.P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY $ex) {
                out = $rh.createExceptionReply ();
                 org.csapi.fw.P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITYHelper.write
                (out, $ex);
        }
        break;
}//end of case 0
case1:
/* org/csapi/fw/fw_access/trust_and_security/IpAPILevelAuthentication/authenticate*/
{
        try {
                byte challenge[] = org.csapi.TpOctetSetHelper.read (in);
                byte $result[] = null;
                $result = this.authenticate (challenge);
                 out = $rh.createReply();
                org.csapi.TpOctetSetHelper.write (out, $result);
        } catch (org.csapi.TpCommonExceptions $ex) {
                out = $rh.createExceptionReply ();
                org.csapi.TpCommonExceptionsHelper.write (out, $ex);
        } catch (org.csapi.fw.P_ACCESS_DENIED $ex) {
                out = $rh.createExceptionReply ();
                org.csapi.fw.P_ACCESS_DENIEDHelper.write (out, $ex);
        }
        break;
}//end of case1
case2:
/* org/csapi/fw/fw_access/trust_and_security/IpAPILevelAuthentication/abortAuthentication*/
{
        try {
                this.abortAuthentication ();
                out = $rh.createReply();
        } catch (org.csapi.TpCommonExceptions $ex) {
```

```
                out = $rh.createExceptionReply ();
                org.csapi.TpCommonExceptionsHelper.write (out, $ex);
        } catch (org.csapi.fw.P_ACCESS_DENIED $ex) {
                out = $rh.createExceptionReply ();
                org.csapi.fw.P_ACCESS_DENIEDHelper.write (out, $ex);
        }
        break;
}//end of case2
case3:
/* org/csapi/fw/fw_access/trust_and_security/IpAPILevelAuthentication/
*authenticationSucceeded
*/
{
        try {
                this.authenticationSucceeded ();
                out = $rh.createReply();
        } catch (org.csapi.TpCommonExceptions $ex) {
                out = $rh.createExceptionReply ();
                org.csapi.TpCommonExceptionsHelper.write (out, $ex);
        } catch (org.csapi.fw.P_ACCESS_DENIED $ex) {
                out = $rh.createExceptionReply ();
                org.csapi.fw.P_ACCESS_DENIEDHelper.write (out, $ex);
        }
        break;
}//end of case3
case4:
/*org/csapi/fw/fw_access/trust_and_security/IpAPILevelAuthentication/
*selectAuthenticationMechanism
*/
{
        try {
                String authMechanismList = org.csapi.fw.TpAuthMechanismListHelper.read (in);
                String $result = null;
                $result = this.selectAuthenticationMechanism (authMechanismList);
                out = $rh.createReply();
                out.write_string ($result);
        } catch (org.csapi.TpCommonExceptions $ex) {
                out = $rh.createExceptionReply ();
                org.csapi.TpCommonExceptionsHelper.write (out, $ex);
        } catch (org.csapi.fw.P_ACCESS_DENIED $ex) {
                out = $rh.createExceptionReply ();
                org.csapi.fw.P_ACCESS_DENIEDHelper.write (out, $ex);
        } catch (org.csapi.fw.P_NO_ACCEPTABLE_AUTHENTICATION_MECHANISM
        $ex){
                out = $rh.createExceptionReply();
                org.csapi.fw.P_NO_ACCEPTABLE_AUTHENTICATION_MECHANISMHelper.
                write(out, $ex);
        }
        break;
}//end of case4
```

```
            case5:
            /* org/csapi/fw/fw_access/trust_and_security/IpAPILevelAuthentication/challenge*/
            {
                try {
                        byte challenge[] = org.csapi.TpOctetSetHelper.read (in);
                        byte $result[] = null;
                        $result = this.challenge (challenge);
                        out = $rh.createReply();
                        org.csapi.TpOctetSetHelper.write (out, $result);
                } catch (org.csapi.TpCommonExceptions $ex) {
                        out = $rh.createExceptionReply ();
                        org.csapi.TpCommonExceptionsHelper.write (out, $ex);
                } catch (org.csapi.fw.P_ACCESS_DENIED $ex) {
                        out = $rh.createExceptionReply ();
                        org.csapi.fw.P_ACCESS_DENIEDHelper.write (out, $ex);
                }
                break;
            }//end of case5
            case6:
            /* org/csapi/fw/fw_access/trust_and_security/IpAuthentication/requestAccess*/
            {
                try {
                        String accessType = org.csapi.fw.TpAccessTypeHelper.read (in);
                        org.csapi.IpInterface clientAccessInterface = org.csapi.IpInterfaceHelper.read (in);
                        org.csapi.IpInterface $result = null;
                        $result = this.requestAccess (accessType, clientAccessInterface);
                        out = $rh.createReply();
                        org.csapi.IpInterfaceHelper.write (out, $result);
                } catch (org.csapi.TpCommonExceptions $ex) {
                        out = $rh.createExceptionReply ();
                        org.csapi.TpCommonExceptionsHelper.write (out, $ex);
                } catch (org.csapi.fw.P_ACCESS_DENIED $ex) {
                        out = $rh.createExceptionReply ();
                        org.csapi.fw.P_ACCESS_DENIEDHelper.write (out, $ex);
                } catch (org.csapi.fw.P_INVALID_ACCESS_TYPE $ex) {
                        out = $rh.createExceptionReply ();
                        org.csapi.fw.P_INVALID_ACCESS_TYPEHelper.write (out, $ex);
                } catch (org.csapi.P_INVALID_INTERFACE_TYPE $ex) {
                        out = $rh.createExceptionReply ();
                        org.csapi.P_INVALID_INTERFACE_TYPEHelper.write (out, $ex);
                }
                break;
            }//end of case6
            default:
                throw new org.omg.CORBA.BAD_OPERATION
                (0, org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);
        }//end of switch
        return out;
    }//end of _invoke( )
```

```
        //Type-specific CORBA::Object operations
        private static String[] __ids =
        {
                "IDL:org/csapi/fw/fw_access/trust_and_security/IpAPILevelAuthentication:1.0",
                "IDL:org/csapi/fw/fw_access/trust_and_security/IpAuthentication:1.0",
                "IDL:org/csapi/IpInterface:1.0"
        };
        public String[] _all_interfaces (org.omg.PortableServer.POA poa, byte[] objectId)
        {
                return (String[])__ids.clone ();
        }
        public IpAPILevelAuthentication _this()
        {
                return IpAPILevelAuthenticationHelper.narrow(
                super._this_object());
        }
        public IpAPILevelAuthentication _this(org.omg.CORBA.ORB orb)
        {
                return IpAPILevelAuthenticationHelper.narrow(super._this_object(orb));
        }
}//class IpAPILevelAuthenticationSkeleton

/**
* Name: IpAPILevelAuthenticationImpl.java
* This is the servant file which implementes the function defined in the idl file.
*/

package com.fwimpl;
import org.csapi.*;
import org.csapi.fw.*;
import org.csapi.fw.fw_access.trust_and_security.*;
import java.sql.*;
import java.net.URL;
import java.lang.*;
import java.util.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.io.*;
import java.security.*;
public class IpAPILevelAuthenticationImpl extends
        org.csapi.fw.fw_access.trust_and_security.IpAPILevelAuthenticationSkeleton
{
        private ORB orb;
          /*prefix string is used to located the CORBA Service
          *its format as follows: corbaname::host_address:port_num#
          */
        private String prefix;
```

```
/*In our implementation , the jdbc driver is mysql driver, so the url string:
  *its format as follows: jdbc:mysql://mysql_server_address:mysql_server_port/
  */
private byte[] fw_digest;
private String url ;
private String login ;              // use your login here
private String password ;                   // use your password here
private boolean access_permission = false;
private InitConfig tool;
private String currentClient;
private String currentPassword;
/*This variable is used to store the callback reference passed by the AS CORBA Client,
  *and this callback reference is used by FW CORBA Client to authenticate the AS CORBA Client
  */
private IpInterface ipclientAPIlevelAuthentication=null;
private boolean alreadyAuth=false;

//public constructor function
public IpAPILevelAuthenticationImpl(ORB orb, IpInterface ipClientAPILevelAuthentication)
{
        System.out.println("IpAPILevelAuthenticationImpl constructor
                                (orb, IpClientAPILevelAuthentications)");
        this.orb=orb;
        this.ipclientAPIlevelAuthentication = ipClientAPILevelAuthentication;
        tool = new InitConfig();
        this.prefix = tool.getPrefix();
        this.url = tool.getSQLURL();
        this.login = tool.getSQLUserName();
        this.password = tool.getSQLPasswd();
}//end of constructor
private static byte[] generateDigest(String username, String password) throws IOException,
        NoSuchAlgorithmException
{
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        DataOutputStream dataout = new DataOutputStream(out);
        long t1 = (new Date()).getTime();
        double q1 = Math.random();
        byte[] protected1 = Protection.makeDigest(username, password, t1, q1);
        dataout.writeUTF(username);
        dataout.writeLong(t1);
        dataout.writeDouble(q1);
        dataout.writeInt(protected1.length);
        dataout.write(protected1);
        dataout.flush();
        //TpOctetSet returnValue = TpOctetSetTool.ByteArrayToTpOCtetSet(out.toByteArray());
        byte[] returnValue = out.toByteArray();
        System.out.println("byte[] size="+out.size());
        return returnValue;
}//end of generateDigest( )
```

```
public String selectEncryptionMethod (String encryptionCaps)
{
        //This function is deprecated, and the selectAuthenticationMechanism () function is suggested to use.
        System.out.println("This request is redirected to the function selectEncryptionMethod()");
        return selectAuthenticationMechanism(encryptionCaps);
}
public String selectAuthenticationMechanism (String authMechanismList)
{
        System.out.println(authMechanismList.toString());
        //This is the default mechanism used in our implementation
        if(authMechanismList.equals("P_OSA_HMAC_SHA1_96")){
                return authMechanismList;
        }
        else{
                System.out.println("The Authentication Mechanism :"+authMechanismList+" provided by
                 the application is not accepted in our implementation!");
                throw new P_NO_ACCEPTABLE_AUTHENTICATION_MECHANISM();
        }
}
public byte[] authenticate (byte[] challenge)
{
        //This function is deprecated, and the challenge() function is suggested to use.
        System.out.println("This request is redirected to the function challenge()");
        return challenge(challenge);
}
public byte[] makeSecretDigest(String username, String password, byte[] random_num){
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        md.update(user.getBytes());
        md.update(password.getBytes());
        md.uodate(random_num);
        return md.digest();
}
public byte[] generateResponse(String username, byte digest[]) throws Exception
{
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        DataOutputStream dataout = new DataOutputStream(out);
        dataout.writeUTF(username);
        dataout.writeInt(digest.length);
        dataout.write(digest);
        dataout.flush();
        return out.toByteArray();
}
public byte[] challenge (byte[] challenge)
{
        //This method is called by the AS to challenge the FW
        System.out.println("Received the Client's challenge " + challenge.toString());
        //The authenticated's FW name and password are known by the AS, so the AS can validate the FW
        String FWName="NCTU_FW";
        String FWPasswd="NCTU_FW";
```

```
        /*The random number field of challenge[]'s (this is so called "Identifier defined in CHAP
        *Section 4.2; and the Response Identifier MUST be copied from the Identifier field " of the
        *Challenge which caused the Response)
        */
        try{
                ByteArrayInputStream in = new ByteArrayInputStream(challenge);
                DataInputStream datain = new DataInputStream(in);
                int msg_len = datain.readInt();
                byte random_num[] = new byte[msg_len];
                datain.readFully(random_num);

                //generate the fw_digest for response
                byte[] fw_secret_digest = makeSecretDigest(FWName,FWPassword,random_num);
                fw_digest = generateResponse(FWName,fw_secret_digest);
                return fw_digest;
        }catch(IOException ioerr){
                System.out.println("In challenge():Generate Respons Error!");
                ioerr.printStackTrace();
                return null;
        }
}
public void abortAuthentication ()
{
        System.out.println("Received the abortAuthentication() message");
        access_permission = false;
}
public void authenticationSucceeded ()
{
        System.out.println("Received the authenticationSucceeded() Message");
        FWClient fw_client = new FWClient(this);
        fw_client.start();
}
public org.csapi.IpInterface requestAccess (String accessType, org.csapi.IpInterface clientAccessInterface)
{
        System.out.println("Received the Accesstype: " + accessType);
        if(!access_permission ){
                System.out.println("This AS is not authenticated!");
                throw new P_ACCESS_DENIED();
        }
        else{
                try
                {
                        IpInterface ipaccessimpl=new IpAccessImpl(orb);
                        return ipaccessimpl;
                }
                catch(Exception e)
                {
                        System.out.println("Get IpAccess servant Failed!");
                        throw new P_ACCESS_DENIED();
                }
```

```
            }
        }
}//class IpAPILevelAuthenticationImpl

/**
* Name: FWClient.java
* This is the FW Authentication CORBA Client.
*/

package com.fwimpl;
import java.io.*;
import java.security.*;
public class FWClient extends Thread
{
        private IpAPILevelAuthenticationImpl ipAPIlevelAuthenticationImpl;
        private IpClientAPILevelAuthenticationImpl ipclientAPIlevelAuthenticationImpl;
        //time and rand_seed to determine the random value
        private long time;
        private double math_seed;
        private byte[] random_num;
        public FWClient(IpAPILevelAuthenticationImpl ref, IpClientAPILevelAuthenticationImpl client_ref)
        {
                ipAPIlevelAuthenticationImpl = ref;
                ipclientAPIlevelAuthenticationImpl = client_ref;
        }
        public byte[] makeBytes (long t, double q)
        {
                try {
                        ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
                        DataOutputStream dataOut = new DataOutputStream(byteOut);
                        dataOut.writeLong(t);
                        dataOut.writeDouble(q);
                        return byteOut.toByteArray();
                }
                catch (IOException e) {
                        return new byte[0];
                }
        }
        public byte[] generateRandNum(long t, double q)
        {
                random_num = makeBytes(t,q);
                return random_num;
        }
        public byte[] generateChallenge(long t, double q) throws Exception
        {
                ByteArrayOutputStream out = new ByteArrayOutputStream();
                DataOutputStream dataout = new DataOutputStream(out);
                random_num = generateRandNum(t,q);
                dataout.writeInt(random_num.length);
                dataout.write(random_num);
```

```
        dataout.flush();
        return out.to toByteArray();
}
public byte[] makeSecretDigest(String username, String password, byte[] _random_num)
{
        MessageDigest md = MessageDigest.getInstance(“SHA-1");
        md.update(user.getBytes());
        md.update(password.getBytes());
        md.uodate(_random_num);
        return md.digest();
}
public String getAccountPassword(String username)
{
        Connection conn;
        try {
                //using mysql driver!
                Class.forName("org.gjt.mm.mysql.Driver").newInstance();
                DriverManager.setLoginTimeout(10);
                conn = DriverManager.getConnection(url,login,password);
                conn.setCatalog( "ServiceDB" );
                Statement st = conn.createStatement();
                ResultSet rs = st.executeQuery(
                "SELECT Passwd FROM AuthClientTable WHERE Client='"+username+"'");
                if(rs.next()){
                        password = rs.getString(1);
                        System.out.println("currentPassword of "+username+" = "+password);
                }
                st.close();
                conn.close();
                return password;
        }catch (Exception e){
                System.out.println(“DataBase Error!”);
                e.printStackTrace();
                return null;
        }
}
public void run()
{
        //Set two seed for the generating the random number
        time = (new Date()).getTime();
        math_seed q = Math.random();
        byte sent_challenge[] = generateChallenge(time,math_seed);
        //The data included in client_digest : username and secret digest
        byte client_digest [] = ipclientAPIlevelAuthenticationImpl.challenge(sent_challenge);
        try{
                ByteArrayInputStream in = new ByteArrayInputStream(client_digest);
                DataInputStream datain = new DataInputStream(in);
                String username = datain.readUTF();
                /*If this AS is authorized by FW, it has the correct password stored in the FW’s database.
                *if this is a faked AS, through it abused someone’s account , and got the random number in the
```

```
                    * flight, but it didn't known the correct password of this account !
                    */
                    //Find the password of this account
                    String password = getAccountPassword(username);
                    //Get the secret digest of the response "client_digest"
                    byte [] client_secret_digest = new byte[datain.readInt()];
                    datain.readFully(client_secret_digest);
                    //Calculating the local digest
                    byte []local_digest = makeSecretDigest(username,password, random_num);
                    //Compare the two results
                    if(MessageDigest.isEqual(client_secret_digest, local_digest))
                    {
                        System.out.println("This is an authorized AS!");
                        ipAPIlevelAuthenticationImpl.access_permission = true;
                        //Notify the AS!
                        ipclientAPIlevelAuthenticationImpl.authenticationSucceeded();
                    }
                    else{
                        System.out.println("This is a faked AS!);
                        ipAPIlevelAuthenticationImpl.access_permission = false;
                    }
                }catche(IOException ioerr){
                    ioerr.printStackTrace();
                }
        }
}//class FWClient

/**
* Name: _IpClientAPILevelAuthenticationStub.java
* org/csapi/fw/fw_access/trust_and_security/_IpClientAPILevelAuthenticationStub.java .
* Generated by the IDL-to-Java compiler (portable), version "3.1"
* idlj –fall –skeletonName %Skeleton fw_if_access.idl
* from fw_if_access.idl
*/

package org.csapi.fw.fw_access.trust_and_security;
public class _IpClientAPILevelAuthenticationStub extends org.omg.CORBA.portable.ObjectImpl implements
        org.csapi.fw.fw_access.trust_and_security.IpClientAPILevelAuthentication
{
        public byte[] authenticate (byte[] challenge)
        {
                org.omg.CORBA.portable.InputStream $in = null;
                try {
                        org.omg.CORBA.portable.OutputStream $out = _request ("authenticate", true);
                        org.csapi.TpOctetSetHelper.write ($out, challenge);
                        $in = _invoke ($out);
                        byte $result[] = org.csapi.TpOctetSetHelper.read ($in);
                        return $result;
                }catch (org.omg.CORBA.portable.ApplicationException $ex) {
                        $in = $ex.getInputStream ();
```

```
            String _id = $ex.getId ();
            throw new org.omg.CORBA.MARSHAL (_id);
        }catch (org.omg.CORBA.portable.RemarshalException $rm) {
            return authenticate (challenge);
        }finally {
            _releaseReply ($in);
        }
}
public void abortAuthentication ()
{
        org.omg.CORBA.portable.InputStream $in = null;
        try {
            org.omg.CORBA.portable.OutputStream $out = _request ("abortAuthentication", true);
            $in = _invoke ($out);
            return;
        }catch (org.omg.CORBA.portable.ApplicationException $ex) {
            $in = $ex.getInputStream ();
            String _id = $ex.getId ();
            throw new org.omg.CORBA.MARSHAL (_id);
        }catch (org.omg.CORBA.portable.RemarshalException $rm) {
            abortAuthentication ( );
        }finally {
            _releaseReply ($in);
        }
}
public void authenticationSucceeded ()
{
        org.omg.CORBA.portable.InputStream $in = null;
        try {
            org.omg.CORBA.portable.OutputStream $out = _request ("authenticationSucceeded", true);
            $in = _invoke ($out);
            return;
        }catch (org.omg.CORBA.portable.ApplicationException $ex) {
            $in = $ex.getInputStream ();
            String _id = $ex.getId ();
            throw new org.omg.CORBA.MARSHAL (_id);
        }catch (org.omg.CORBA.portable.RemarshalException $rm) {
            authenticationSucceeded ( );
        }finally {
            _releaseReply ($in);
        }
}
public byte[] challenge (byte[] challenge)
{
        org.omg.CORBA.portable.InputStream $in = null;
         try {
            org.omg.CORBA.portable.OutputStream $out = _request ("challenge", true);
            org.csapi.TpOctetSetHelper.write ($out, challenge);
            $in = _invoke ($out);
            byte $result[] = org.csapi.TpOctetSetHelper.read ($in);
```

```
                return $result;
            }catch (org.omg.CORBA.portable.ApplicationException $ex) {
                $in = $ex.getInputStream ();
                String _id = $ex.getId ();
                throw new org.omg.CORBA.MARSHAL (_id);
            }catch (org.omg.CORBA.portable.RemarshalException $rm) {
                return challenge (challenge);
            }finally {
                _releaseReply ($in);
            }
        }
        private static String[] __ids = {
            "IDL:org/csapi/fw/fw_access/trust_and_security/IpClientAPILevelAuthentication:1.0",
            "IDL:org/csapi/IpInterface:1.0"
        };
        public String[] _ids ()
        {
            return (String[])__ids.clone ();
        }

        private void readObject (java.io.ObjectInputStream s) throws java.io.IOException
        {
            String str = s.readUTF ();
            String[] args = null;
            java.util.Properties props = null;
            org.omg.CORBA.Object obj = org.omg.CORBA.ORB.init (args, props).string_to_object (str);
            org.omg.CORBA.portable.Delegate delegate =
            ((org.omg.CORBA.portable.ObjectImpl) obj)._get_delegate ();
            _set_delegate (delegate);
        }
        private void writeObject (java.io.ObjectOutputStream s) throws java.io.IOException
        {
            String[] args = null;
            java.util.Properties props = null;
            String str = org.omg.CORBA.ORB.init (args, props).object_to_string (this);
            s.writeUTF (str);
        }
}//class _IpClientAPILevelAuthenticationStub
```

A.3 AS Initial Module

```
/**
* Name: _IpInitialStub.java
* org/csapi/fw/fw_access/trust_and_security/_IpInitialStub.java .
* Generated by the IDL-to-Java compiler (portable), version "3.1"
* idlj –fall –skeletonName %Skeleton fw_if_access.idl
* from fw_if_access.idl
*/

package org.csapi.fw.fw_access.trust_and_security;
```

```
public class _IpInitialStub extends org.omg.CORBA.portable.ObjectImpl implements
        org.csapi.fw.fw_access.trust_and_security.IpInitial
{
        public org.csapi.fw.TpAuthDomain initiateAuthentication (org.csapi.fw.TpAuthDomain clientDomain,
                String authType) throws org.csapi.TpCommonExceptions, org.csapi.fw.P_INVALID_DOMAIN_ID,
                org.csapi.P_INVALID_INTERFACE_TYPE, org.csapi.fw.P_INVALID_AUTH_TYPE
        {
                org.omg.CORBA.portable.InputStream $in = null;
                try {
                        org.omg.CORBA.portable.OutputStream $out = _request ("initiateAuthentication", true);
                        org.csapi.fw.TpAuthDomainHelper.write ($out, clientDomain);
                        org.csapi.fw.TpAuthTypeHelper.write ($out, authType);
                        $in = _invoke ($out);
                        org.csapi.fw.TpAuthDomain $result = org.csapi.fw.TpAuthDomainHelper.read ($in);
                        return $result;
                }catch (org.omg.CORBA.portable.ApplicationException $ex) {
                        $in = $ex.getInputStream ();
                        String _id = $ex.getId ();
                        if (_id.equals ("IDL:org/csapi/TpCommonExceptions:1.0"))
                                throw org.csapi.TpCommonExceptionsHelper.read ($in);
                        else if (_id.equals ("IDL:org/csapi/fw/P_INVALID_DOMAIN_ID:1.0"))
                                throw org.csapi.fw.P_INVALID_DOMAIN_IDHelper.read ($in);
                        else if (_id.equals ("IDL:org/csapi/P_INVALID_INTERFACE_TYPE:1.0"))
                                throw org.csapi.P_INVALID_INTERFACE_TYPEHelper.read ($in);
                        else if (_id.equals ("IDL:org/csapi/fw/P_INVALID_AUTH_TYPE:1.0"))
                                throw org.csapi.fw.P_INVALID_AUTH_TYPEHelper.read ($in);
                        else
                                throw new org.omg.CORBA.MARSHAL (_id);
                }catch (org.omg.CORBA.portable.RemarshalException $rm) {
                        return initiateAuthentication (clientDomain, authType);
                }finally {
                        _releaseReply ($in);
                }
        }
        public org.csapi.fw.TpAuthDomain initiateAuthenticationWithVersion (org.csapi.fw.TpAuthDomain
                clientDomain, String authType, String frameworkVersion) throws org.csapi.TpCommonExceptions,
                org.csapi.fw.P_INVALID_DOMAIN_ID, org.csapi.P_INVALID_INTERFACE_TYPE,
                org.csapi.fw.P_INVALID_AUTH_TYPE, org.csapi.P_INVALID_VERSION
        {
                org.omg.CORBA.portable.InputStream $in = null;
                try {
                        org.omg.CORBA.portable.OutputStream $out =
                        _request ("initiateAuthenticationWithVersion", true);
                        org.csapi.fw.TpAuthDomainHelper.write ($out, clientDomain);
                        org.csapi.fw.TpAuthTypeHelper.write ($out, authType);
                        org.csapi.TpVersionHelper.write ($out, frameworkVersion);
                        $in = _invoke ($out);
                        org.csapi.fw.TpAuthDomain $result = org.csapi.fw.TpAuthDomainHelper.read ($in);
                        return $result;
                }catch (org.omg.CORBA.portable.ApplicationException $ex) {
```

```
                $in = $ex.getInputStream ();
                String _id = $ex.getId ();
                if (_id.equals ("IDL:org/csapi/TpCommonExceptions:1.0"))
                        throw org.csapi.TpCommonExceptionsHelper.read ($in);
                else if (_id.equals ("IDL:org/csapi/fw/P_INVALID_DOMAIN_ID:1.0"))
                        throw org.csapi.fw.P_INVALID_DOMAIN_IDHelper.read ($in);
                else if (_id.equals ("IDL:org/csapi/P_INVALID_INTERFACE_TYPE:1.0"))
                        throw org.csapi.P_INVALID_INTERFACE_TYPEHelper.read ($in);
                else if (_id.equals ("IDL:org/csapi/fw/P_INVALID_AUTH_TYPE:1.0"))
                        throw org.csapi.fw.P_INVALID_AUTH_TYPEHelper.read ($in);
                else if (_id.equals ("IDL:org/csapi/P_INVALID_VERSION:1.0"))
                        throw org.csapi.P_INVALID_VERSIONHelper.read ($in);
                else
                        throw new org.omg.CORBA.MARSHAL (_id);
        }catch (org.omg.CORBA.portable.RemarshalException $rm) {
                return initiateAuthenticationWithVersion (clientDomain, authType, frameworkVersion);
        }finally {
                _releaseReply ($in);
        }
    }
    private static String[] __ids = {
        "IDL:org/csapi/fw/fw_access/trust_and_security/IpInitial:1.0",
        "IDL:org/csapi/IpInterface:1.0"
    };
    public String[] _ids ()
    {
        return (String[])__ids.clone ();
    }
    private void readObject (java.io.ObjectInputStream s) throws java.io.IOException
    {
        String str = s.readUTF ();
        String[] args = null;
        java.util.Properties props = null;
        org.omg.CORBA.Object obj = org.omg.CORBA.ORB.init (args, props).string_to_object (str);
        org.omg.CORBA.portable.Delegate delegate =
        ((org.omg.CORBA.portable.ObjectImpl) obj)._get_delegate ();
        _set_delegate (delegate);
    }
    private void writeObject (java.io.ObjectOutputStream s) throws java.io.IOException
    {
        String[] args = null;
        java.util.Properties props = null;
        String str = org.omg.CORBA.ORB.init (args, props).object_to_string (this);
        s.writeUTF (str);
    }
}//class _IpInitialStub
```

A.4 AS Authentication Module

```
/**
* Name: IpClientAPILevelAuthenticationSkeleton.java
* org/csapi/fw/fw_access/trust_and_security/IpClientAPILevelAuthenticationSkeleton.java .
* Generated by the IDL-to-Java compiler (portable), version "3.1"
* idlj –fall –skeletonName %Skeleton fw_if_access.idl
* from fw_if_access.idl
*/

package org.csapi.fw.fw_access.trust_and_security;
public abstract class IpClientAPILevelAuthenticationSkeleton extends org.omg.PortableServer.Servant
        implements org.csapi.fw.fw_access.trust_and_security.IpClientAPILevelAuthenticationOperations,
        org.omg.CORBA.portable.InvokeHandler
{
        private static java.util.Hashtable _methods = new java.util.Hashtable ();
        static
        {
                _methods.put ("authenticate", new java.lang.Integer (0));
                _methods.put ("abortAuthentication", new java.lang.Integer (1));
                _methods.put ("authenticationSucceeded", new java.lang.Integer (2));
                _methods.put ("challenge", new java.lang.Integer (3));
        }
        public org.omg.CORBA.portable.OutputStream _invoke (String $method,
                org.omg.CORBA.portable.InputStream in, org.omg.CORBA.portable.ResponseHandler $rh)
        {
                org.omg.CORBA.portable.OutputStream out = null;
                java.lang.Integer __method = (java.lang.Integer)_methods.get ($method);
                if (__method == null)
                        throw new org.omg.CORBA.BAD_OPERATION (0,
                        org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);
                switch (__method.intValue ())
                {
                        case 0:
                        /*org/csapi/fw/fw_access/trust_and_security/IpClientAPILevelAuthentication/authenticate*/
                        {
                                byte challenge[] = org.csapi.TpOctetSetHelper.read (in);
                                byte $result[] = null;
                                $result = this.authenticate (challenge);
                                out = $rh.createReply();
                                org.csapi.TpOctetSetHelper.write (out, $result);
                                break;
                        }//end of case 0
                        case 1:
                        /*org/csapi/fw/fw_access/trust_and_security/IpClientAPILevelAuthentication/
                        * abortAuthentication
                        */
                        {
                                this.abortAuthentication ();
                                out = $rh.createReply();
```

```
                    break;
                }//end of case1
                case 2:
                /* org/csapi/fw/fw_access/trust_and_security/IpClientAPILevelAuthentication/
                *authenticationSucceeded
                */
                {
                    this.authenticationSucceeded ();
                    out = $rh.createReply();
                    break;
                }//end of case2
                case3:
                /* org/csapi/fw/fw_access/trust_and_security/IpClientAPILevelAuthentication/challenge*/
                {
                    byte challenge[] = org.csapi.TpOctetSetHelper.read (in);
                    byte $result[] = null;
                    $result = this.challenge (challenge);
                    out = $rh.createReply();
                    org.csapi.TpOctetSetHelper.write (out, $result);
                    break;
                }//end of case3
                default:
                    throw new org.omg.CORBA.BAD_OPERATION (0,
                    org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);
            }
            return out;
        }//end of _invoke()
        private static String[] __ids = {
            "IDL:org/csapi/fw/fw_access/trust_and_security/IpClientAPILevelAuthentication:1.0",
            "IDL:org/csapi/IpInterface:1.0"
        };
        public String[] _all_interfaces (org.omg.PortableServer.POA poa, byte[] objectId)
        {
            return (String[])__ids.clone ();
        }
        public IpClientAPILevelAuthentication _this()
        {
            return IpClientAPILevelAuthenticationHelper.narrow(super._this_object());
        }
        public IpClientAPILevelAuthentication _this(org.omg.CORBA.ORB orb)
        {
            return IpClientAPILevelAuthenticationHelper.narrow(super._this_object(orb));
        }
}//class IpClientAPILevelAuthenticationSkeleton
```

```
/**
* Name: IpClientAPILevelAuthenticationImpl.java
* This is the servant file which implementes the function defined in the idl file.
*/

package com.asimpl;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.*;
import java.sql.*;
import java.io.*;
import java.security.*;
public class IpClientAPILevelAuthenticationImpl extends
        org.csapi.fw.fw_access.trust_and_security.IpClientAPILevelAuthenticationSkeleton
{
        private ORB orb;
        private String currentClient;
        private String currentPassword;
        public IpClientAPILevelAuthenticationImpl(ORB orb, String usrname, String passwd)
        {
                this.orb=orb;
                this.currentClient = usrname;
                this.currentPassword = passwd;
        }
        public byte[] authenticate (byte[] challenge)
        {
                System.out.println(“This function is already deprecated!”);
                System.out.println(“Redirecting to challenge() function”);
                return challenge(challenge);
        }
        public void abortAuthentication ()
        {
                System.out.println("AbortAuthentication");
        }
        public void authenticationSucceeded ()
        {
                System.out.println("authenticationSucceeded");
        }
        public byte[] makeSecretDigest(String username, String password, byte[] random_num){
                MessageDigest md = MessageDigest.getInstance(“SHA-1");
                md.update(user.getBytes());
                md.update(password.getBytes());
                md.uodate(random_num);
                return md.digest();
        }
        public byte[] generateResponse(String username, byte digest[]) throws Exception
        {
```

```
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            DataOutputStream dataout = new DataOutputStream(out);
            dataout.writeUTF(username);
            dataout.writeInt(digest.length);
            dataout.write(digest);
            dataout.flush();
            return out.toByteArray();
        }
        public byte[] challenge (byte[] challenge)
        {
            try{
                ByteArrayInputStream in = new ByteArrayInputStream(challenge);
                DataInputStream datain = new DataInputStream(in);
                int msg_len = datain.readInt();
                byte random_num[] = new byte[msg_len];
                datain.readFully(random_num);
                //generate the client_digest for response
                byte[] client_secret_digest = makeSecretDigest(currentClient,curentPassword,random_num);
                client_digest = generateResponse(currentClient,client_secret_digest);
                return fw_digest;
            }catch(IOException ioerr){
                System.out.println("In challenge():Generate Respons Error!");
                ioerr.printStackTrace();
                return null;
            }
        }

}//class IpClientAPILevelAuthenticationImpl
```

```
/**
* Name: ASClient.java
* This is the AS Authentication CORBA Client.
* This class is used by AppLogic
*/
package com.asimpl;
import org.csapi.*;
import org.csapi.fw.*;
import org.csapi.fw.fw_access.trust_and_security.*;
import org.csapi.fw.fw_application.discovery.*;
import org.csapi.fw.fw_application.service_agreement.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import java.util.*;
import java.io.*;
import java.security.*;
public class ASClient
{
        private ORB orb;
```

```
private AppLogic appLogic;
private POA rootpoa;
private IpInitial ipinitialimpl;
private IpAPILevelAuthentication ipAPIlevelauthenticationimpl;
private TpAuthDomain cliDomain=null;
private TpAuthDomain fwDomain=null;
private String authType="P_OSA_AUTHENTICATION";
private String version="FWv1.0";
private byte[] random_num;
public ASClient(AppLogic _appLogic, ORB _orb)
{
        orb = _orb;
        appLogic = _appLogic;
        try{
                rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
                rootpoa.the_POAManager().activate();
        }catch(Exception e){
                System.out.println("Get rootpoa Failed!")
        }
}
public byte[] makeBytes (long t, double q)
{
        try {
                ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
                DataOutputStream dataOut = new DataOutputStream(byteOut);
                dataOut.writeLong(t);
                dataOut.writeDouble(q);
                return byteOut.toByteArray();
        }
        catch (IOException e) {
                return new byte[0];
        }
}
public byte[] generateRandNum(long t, double q)
{
        random_num = makeBytes(t,q);
        return random_num;
}
public byte[] generateChallenge(long t, double q) throws Exception
{
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        DataOutputStream dataout = new DataOutputStream(out);
        random_num = generateRandNum(t,q);
        dataout.writeInt(random_num.length);
        dataout.write(random_num);
        dataout.flush();
        return out.to toByteArray();
}
public byte[] makeSecretDigest(String username, String password, byte[] _random_num)
{
```

```
        MessageDigest md = MessageDigest.getInstance(“SHA-1");
        md.update(user.getBytes());
        md.update(password.getBytes());
        md.uodate(_random_num);
        return md.digest();
}
public void startInitialAccess(String username, String password)
{
        IpClientAPILevelAuthenticationImpl ipclientAPIlevelAuthenticationimpl =
        new IpClientAPILevelAuthenticationImpl(orb, usrname, passwd);
        //This is the reference for the using of the FW’s callback
        IpClientAPILevelAuthentication ipclientAPIlevelauthentication_ref = null;
        try{
                org.omg.CORBA.Object ref =
                rootpoa.servant_to_reference(ipclientAPIlevelAuthenticationimpl);
                ipclientAPIlevelauthentication _ref = IpClientAPILevelAuthenticationHelper.narrow(ref);
        }catch (Exception e) {
                System.out.println(“IpClientAPILevelAuthentication : servant_to_reference() error!”);
                e.printStackTrace();
        }
        try{
                TpDomainID cliDomainID = new TpDomainID();
                cliDomainID.ClientAppID("CCL_OSA_Client");
                cliDomain = new TpAuthDomain(cliDomainID, ipclientAPIlevelauthentication_ref);
                org.omg.CORBA.Object obj =
                orb.string_to_object("corbaname::pcs.csie.nctu.edu.tw:3500#IpInitial");
                //get the reference for IpInitial Interface
                ipinitialimpl = IpInitialHelper.narrow(obj);
                // 1. initiateAuthenticationWithVersion()
                fwAuthDomain =
                ipinitialimpl.initiateAuthenticationWithVersion(cliDomain, authType, version);
                if(fwAuthDomain!=null)
                        System.out.println("FrameWork domainID="+fwAuthDomain.DomainID.FwID());
                //get the reference of the IpAPILevelAuthentication interface
                ipAPIlevelauthenticationimpl =
                IpAPILevelAuthenticationHelper.narrow(fwAuthDomain.AuthInterface);
                // 2. selectAuthenticationMechanism()
                ipAPIlevelauthenticationimpl. selectAuthenticationMechanism("P_OSA_HMAC_SHA1_96");
                // 3. challenge
                long t = (new Date()).getTime();
                double q = Math.random();
                byte[] sent_challenge = generateChallenge(t,q);
                byte[] fw_digest = ipapilevelauthenticationimpl.challenge(sent_challenge);
                ByteArrayInputStream in = new ByteArrayInputStream(fw_digest);
                DataInputStream datain = new DataInputStream(in);
                String username = datain.readUTF();
                /*If this FW is authorized by AS, it has the correct password stored in the AS’s List.
                *if this is a faked FW, through it abused someone’s account , and got the random number in the
                * flight, but it didn’t known the correct password of this account !
                */
```

```
                //The only FW is authenticated by the AS is NCTU_CCL_FW
                String password = “Pass_NCTU_CCL_FW”;
                //Get the secret digest of the response “fw_digest”
                byte [] fw_secret_digest = new byte[datain.readInt()];
                datain.readFully(fw_secret_digest);
                //Calculating the local digest
                byte []local_digest = makeSecretDigest(username,password, random_num);

                //Compare the two results
                if(MessageDigest.isEqual(fw_secret_digest, local_digest))
                {
                        System.out.println("This is an authorized FW!");
                        // 4. authenticationSucceeded()
                        ipAPIlevelauthenticationimpl.authenticationSucceeded();
                        appLogic.requestaccee_permission = true;
                }
                else{
                        System.out.println("This is a faked FW!);
                        appLogic.requestaccee_permission = false;
                }
        }catch(Exception err){
                System.out.println(“Error in startInitialAccess()”);
                err.printStackTrace();
        }
}//class ASClient
```