ADAPTABLE SERVICES AND APPLICATIONS FOR NETWORKS

Josep Polo and Jaime Delgado
Universitat Pompeu Fabra. Technology Department, Psg. de Circumval·lació, 8, 08003 Barcelona, Spain {josep.polo, jaime.delgado}@upf.edu, http://dmag.upf.edu/

Abstract: New services and applications that use extensively telecommunication networks are currently developed. They need an open access to the telecommunication networks for adapting to them. An API (Application Programming Interface) that permits this objective is the Parlay/OSA API. Operators offer tools that facilitate the development of applications and services, but these tools are different, so it is not possible to develop a unique application for different operators. To solve this great inconvenience, we propose a solution to permit interoperability among different development tools.

Key words: Parlay; OSA; network; application; service.

1. INTRODUCTION

The use of network services is growing. In order to facilitate their development an API that offers many advantages is Parlay and Open Service Access (OSA). This API facilitates the services development, but it is still very complex, difficult and hard to use. Many operators and service providers offer different development tools, SDKs (Software Development Kit), that facilitate the development and implementation phases.

This document begins with an introduction about the Parlay/OSA (P/OSA) API and it shows its complexity. Then, it demonstrates how a service can be developed using those tools more easily that directly with the API. We show one application for each SDK. Later, we propose a solution to interoperability among different SDKs. We suggest a method to develop a unique service to be used in several SDKs, and finally we obtain some conclusions.

2. PARLAY/OSA OVERVIEW

Parlay and OSA (P/OSA) are an open API for communications networks. This API can support present and future networks. It provides a layer of abstraction for service developers. It enables telecom operators and service providers to offer the same services for all existing underlying networks: mobile, fixed and IP networks, without adapting the application to network specific protocols.

This API permits to obtain network-related context information, it can facilitate value-added services development, it can make network communications simpler and powerful, independently of which type of network.

2.1 Description

The P/OSA model is split into three main entities [12]:

- The client application: the application developed by a third party can access to the network features through the P/OSA interface.
- The framework: it offers support functions. For instance, security, integrity and management features.
- The services: they offer access to network features. For instance mobility, messaging, terminal management, user interaction and call control.

P/OSA has several interfaces to allow access to different network functionalities or services [12]: Framework, Call control, Data session control, User interaction, Mobility, Generic messaging, Terminal capabilities, Connectivity management, Account management, Charging, Policy management, Presence and availability management.

2.2 Application architecture

When developing a service to use P/OSA, it has three main steps:

- Authentication: before the application can use the network services, the application and the framework authenticate each other. Authentication prevents unauthorized access to the services and permits to determine the privileges and permissions to the application.
- Service selection: after the authentication phase, the application can select the service interface to use. It is usual to sign an agreement before use of the interface. The application can select the service to use.
- Service use: if the previous phases have concluded properly, the application can use the selected service.

To obtain a general idea on interface use, we shall explain only the first step. This step is common for all developing services or applications and before it can use any network service there must be an authentication step.

This step has several phases:

- The application performs an initial contact to the framework. In this first contact the application requests authentication. The framework answers with the authentication to be used.
- Then the application requests authentication from the framework.
- After the previous authentication, the framework requests authentication from the application.
- When the application and framework have authenticated each other, the application can select the service interface to use. This corresponds to the service selection step.

These phases correspond to many operations. The authentication step is a quite simple operation. The other steps are usually more complicated and large, depending on the service.

The previous authentication step description is enough to show a good idea of the use of P/OSA. From the previous paragraphs it can be concluded that the direct use of P/OSA can be very complicated, tedious, difficult and long effort. To solve this situation, few libraries exist. They shield the details of P/OSA use. They provide abstraction from the original API. Usually, these libraries are developed to use Java APIs.

3. DEVELOPMENT TOOLS

Many operators have developed tools that facilitate the effort to create applications. Usually, these tools are composed by a SDK and a simulator.

Those SDKs and simulators share characteristics. The most important are:

- Java software libraries. P/OSA defines a language-independent API and they use the Object Management Group (OMG) [6] Unified Modelling Language (UML) and Interface Definition Language (IDL) [5] that is based on the OMG's CORBA IDL. Using java libraries permits the abstraction from the CORBA.
- Partial emulation of P/OSA APIs. The systems don't support all P/OSA interfaces. They are enough for developing and testing most applications.
- They can run on an off-line way, no network connection is needed if a simulator is used. This can permit an easy application development.
- They have a similar structure. They offer a Java library, which simplify the development, but the application can use CORBA, if necessary.

To simplicity and avoid confusion, we show only two of them to obtain a general idea of their use. They are: Lucent - MiLife ISG SDK [5] (MI SDK) and Ericsson NRG SDK [1] (EN SDK).

4. SERVICE AND APPLICATION IMPLEMENTATION

Services and applications have three big parts, when using these systems:

- Initialization. In this step the application initializes all needed processes and obtains all resources for the correct operation, and interacts with the framework, which enables an application to obtain and release service managers.
- Main functionality. This is the most important part of the application. In this step the application interacts with the network to do the features for those it is developed. It uses service managers to send requests to the network, and receives responses from the network.
- Finalization. The application stops interacting with the network. The service managers are released and framework access is terminated.

All applications have the first and third phases (initialization and finalization) very similar, but the actually important step is the second one (main functionality). Each application has this phase different.

To show how to implement a service or application using these tools, we explain the main steps necessaries for a simple sample application. We think that it is enough to have a good idea how a service can be developed.

4.1 Sample application

The sample application demonstrates the framework. It obtains and shows available services defined in the network.

The basic steps of each three main parts of this application are:

- Initialization.
  - *Prepare the resources and add components and read the configuration*
  - Initiate the access to the framework
- Main functionality.
  - Obtain the names of all available service types
  - Obtain a service manager
- Finalization.
  - Releases a service manager by terminating its service level agreement
  - Release the system resources obtained in the initialization phase

Following sections show the most important operations to use each SDK.

4.1.1 Lucent - MiLife ISG SDK

We focus on how to use the MI SDK to obtain the desired P/OSA functionality. We only describe the operations that perform those functions.

The interfaces use (see section 2.2) has three main steps: authentication, service selection and service use. The first step, authentication, correspond to the initial operations, before the service use.

The first step can be done using the class `FrameworkAdapterFactory`. A framework adapter can be created using this class. The sentence is:

```
FrameworkAdapterObject =
   FrameworkAdapterInstance.createFrameworkAdapter
                    (validAuthenticationCredentials);
```

The `FrameworkAdapter` is an interface that defines the framework classes. A reference to this interface can be obtained by the previous method. The `validAuthenticationCredentials` are parameters that have been set to proper values.

This single operation performs that first step, authentication. The needed operations, if P/OSA is used directly, are related in section 2.2. It can be seen that this class performs most work and simplifies the application development.

The following operation is to obtain the available services. This can be done using the method `listServices`. The sentence to use is:

```
String[] object =
               FrameworkAdapterObject.listServices();
```

This method returns a list of available services.

The next operation is to obtain the service adapter. This operation can be done using a method specific to the wanted service. For instance, if we want to select the User Location service, the sentence to use is:

```
serviceAdapter =
FrameworkAdapterObject.selectUserLocationService();
```

This service permits to obtain the geographical location of users, but if we want to select the Messaging service, to manage, send or receive messages, the sentence to use is:

```
serviceAdapter =
  FrameworkAdapterObject.selectMessagingService();
```

As we can see, each service is selected using a specific method.

Then the service can be used.

When the services are no longer needed, it is necessary to release resources. This operation begins with the method `destroy`. The sentence to use is: `serviceAdapter.destroy();`

It allows this adapter to clean up when it is no longer used.

The releasing operation continues using the method `endAccess` that releases resources used by the framework. The sentence to use is: `FrameworkAdapterObject.endAccess()`. It ends the access session.

Finally, the method to use is `destroy`. This method allows the `FrameworkAdapter` to clean up when it is no longer used. The sentence to use is: `FrameworkAdapterObject.destroy();`

4.1.2 Ericsson NRG SDK

In this section we focus on how to use the EN SDK to obtain the desired P/OSA functionality.

The first step can be done using the class `FWproxy`. This class is for handling interaction with the framework, which enables an application to obtain and release service managers. The sentence to use is:

```
FwproxyObject = new Fwproxy(configuration);
```

The `configuration` has been set previously to proper values.

The following operation is to obtain the available services. This can be done using the method `listServiceTypes` of the class `FWproxy`. The sentence to use is:

```
String[] object =
                    FwproxyObject.listServiceTypes();
```

The `FwproxyObject` is the previously object created. This operation returns the names of all available service types.

The next operation is to obtain the service manager. This operation can be done using the method `obtainSCF` of the class `FWproxy`. The sentence to use is:

```
IpServiceObject =
                FwproxyObject.obtainSCF(UserLocation);
```

The `UserLocation` is the service to use. This operation returns a service manager. In this example, this service allows application to obtain the geographical location of users. Or if we want to manage, send or receive messages, for instance, then the sentence is:

```
IpServiceObject = FwproxyObject.obtainSCF(Message);
```

When the service is no longer needed, it is necessary to release resources. This operation begins using the method `releaseSCF` of `Fwproxy`. The sentence to use is:

```
FwproxyObject.releaseSCF(IpServiceObject);
```

The releasing operation continues using the method `endAccess` that releases resources used by the framework. It belongs to the class `FWproxy`. The sentence to use is: `FwproxyObject.endAccess();`

The method to releases all resources is `dispose` of the class `FWproxy`.

```
FwproxyObject.dispose();
```

5. INTEROPERABILITY

In the previous sections we have seen that those SDKs are different. Due to this reason, a unique application cannot be used with different SDKs. It is necessary to develop different applications, one for each SDK. Those applications are nearly equal except the use of P/OSA interface through SDK java libraries. To isolate the developing application from the particular SDK used we propose to use an interface that hides the particular implementation of each SDK. This interface is in a preliminary status. In this section we present the first results, that where introduced in [8].

This interface shows to developing application unique classes to access P/OSA interfaces, independently of which library used. In following sections we show a possible implementation of this interface.

This interface can consists in a set of unique classes with different implementations. This can be done in java using abstract classes. The new developing application uses the abstract classes and depending on which SDK we want to use, we utilize an implementation or another. All particular details are encapsulated on each implementation.

Following sections show this interface applied to the framework. There is an abstract framework class and two no abstract framework classes, one for each SDK.

5.1 Framework abstract class

The abstract classes contain a definition for all SDKs. These classes have the methods to access the P/OSA interfaces to be used by the application. Figure 1 shows a possible abstract class for the framework that contains the definition of few methods, it is not complete. They are some of the initialization and finalization steps methods. None of then is implemented, because the implementation depends on the particular SDK used.

```
public abstract class Framework {
    ...
    public abstract String [] listServices (); // list the available services
    ...
    public abstract void endAccess (); // releases resources used by the framework
    public abstract void destroy (); // disposes the framework
}
```

Figure 1. Framework abstract class.

This abstract class defines a neutral framework. As we can see, there is not any constructor. The constructor task will be done by next classes. They create the framework object: a `FrameworkAdapterFactory` (MI SDK) or a `Fwproxy` (EN SDK). This class only contains method definitions that apply on the created object.

It is necessary to define more abstract classes for all those classes that exist in each Java library. For instance, when the framework is successful accessed, next step is to obtain the service manager. In MI SDK is a service dependent class (`UserLocationAdapter`, `MessagingAdapter`, etc.) and in EN SDK is an `IpService`.

5.2 ISG framework class

When using MI SDK we actually want to use `FrameworkAdapter`.

We define a class that inherits from the abstract class all its methods, it implements them and adds the framework constructor. Figure 2 shows a possible implementation of the framework if MI SDK is used.

```
public class ISGFramework extends Framework {

    FrameworkAdapter fwISG; // ISG framework

    public ISGFramework() { // ISG constructor
        FrameworkAdapterFactory.createFrameworkAdapter("Sample","psw");
    }
    ...
    public String [] listServices() {
        return fwISG.listServices(); // list the available ISG services
    }
    ...
    public void endAccess () {
        fwISG.endAccess(); // releases resources used by the ISG framework
    }
    public void destroy () {
        fwISG.destroy(); // disposes the ISG framework
    }
}
```

Figure 2. ISG framework class.

As we can see, all general methods (abstract methods in framework class, Figure 1) use the particular methods defined in MI SDK. Most methods are very simple: they use MI SDK methods directly. For instance `destroy` method uses the `dispose` ISG method. Section 4.1.1 shows the MI SDK methods listed in Figure 2. Not all methods are so simple as shown, but to demonstrate the essence of this interface, this example is enough.

5.3 NRG framework class

```
public class NRGFramework extends Framework {

    FWproxy fwNRG; // NRG framework

    public NRGFramework () { // constructor
        Configuration.INSTANCE.load("configuration.ini");
        fwNRG = new FWproxy(Configuration.INSTANCE); // NRG constructor
    }
    ...
    public String [] listServices() {
        return fwNRG.listServiceTypes(); // list the available NRG services
    }
    ...
    public void endAccess (){
        fwNRG.endAccess(); // releases resources used by the NRG framework
    }
    public void destroy () {
        fwNRG.dispose(); // disposes the NRG framework
    }
}
```

Figure 3. NRG framework class.

When using EN SDK we actually have to use `Fwproxy`. As previous class, we define a class that inherit from the abstract class all its methods and adds the framework constructor. Figure 3 shows a possible implementation of the framework class if EN SDK is used.

5.4 The application

The application corresponds to the developed application. This sample is very simple and only creates the framework, then obtain an available service list, it shows them and finally release all sources. Figure 4 shows this simple application.

```
public static void main(String[] args) {

    Framework fw; // framework object
    String[] serviceList; // list of available services

    if (args[0].equals("NRG")) // discriminates the SDK
        fw = new NRGFramework(); // NRG framework creator
    else
        fw = new ISGFramework(); // ISG framework creator

    serviceList = fw.listServices(); // obtains the service list
    for (int i=0; i < serviceList.length; i++)
        System.out.println("Service =" + serviceList[i]); // shows the services
    fw.endAccess(); // releases resources used by the framework
    fw.destroy(); // disposes the framework
}
```

Figure 4. Sample application.

As we can see, only at the application beginning there is an explicit indication of which SDK we are using. Then, later the application doesn't care about that question. All methods used are those in the abstract class (Figure 1), but when the application runs, the real methods used are those in the ISG framework class (Figure 2) if the MI SDK is used or the NRG framework class (Figure 3) if the NE SDK is used.

6. CONCLUSIONS

The objective of P/OSA is to open an interface network to third parties, to permit developing new applications and services. We have shown that it is very powerful to develop new applications that use the telecommunication networks, although those interfaces shield most of the detail and eliminates most effort, those interfaces are very complex, difficult and hard to use. Several SDKs exist that solve these obstacles. They permit to use the Java language, simplifying the portability of the applications. This fact permits the use of very different terminals and resources, expanding, in a wide way, the use and reuse of the developed applications.

These SDKs have simulators that permit to develop application without accessing to a network. They can be done in a stand-alone way, without the difficulty that can add the fact to access a real network.

Those tools have a disadvantage: they don't support all interfaces defined in P/OSA, but they are enough for developing and test most applications.

We have shown the main steps that are necessary for developing an application: initialization, main functionality and finalization. We have applied those steps to a sample application to two SDKs: Lucent - MiLife ISG SDK and Ericsson NRG SDK. We have developed two sample applications, one for each SDK. They are only different when accessing to P/OSA functionality. So, it is necessary to develop an application for each SDK intended to use. To isolate the application from the particular SDK we propose to use an interface that hides the particular implementation of each SDK.

This interface is composed by a set of abstract classes that define a generic use of P/OSA and different implementations, depending on which SDKs want to be used. We have shown a partial implementation of this interface and we have applied to a sample application to show their structure and use. The whole interface is more complicated than that shown in this document, but this one is enough to show its philosophy and the way to develop applications.

This is only a preliminary work. We are currently working to expand these results: extend the interface to the whole SDK and to cover more SDKs. It is expected that few parts of SDKs will be more difficult to implement than others, and that few SDKs will be more complex than others. Due to this reason, we cannot assure that we will be able to apply this interface to all existing SDKs. We are currently working on these problems and we will report our results in the future.

ACKNOWLEDGEMENTS

This work has been partly supported by the Spanish administration (AgentWeb project, TIC 2002-01336) and is being developed within VISNET (IST-2003-506946, www.visnet-noe.org), a European Network of Excellence, funded under the European Commission IST FP6 program.

REFERENCES

1. Ericsson NRG Simulator and Software Development Kit (SDK) http://www.ericsson.com/mobilityworld/sub/open/technologies/parlay/tools/parlay_sdk
2. ETSI ES 202 915-1 V1.2.1 (2003-08) Open Service Access (OSA); Application Programming Interface (API) http://www.parlay.org/specs/index.asp
3. MiLife™ ISG SDK 3.0 Documentation. http://www.lucent.com/products/solution/0,,CTID+2019-STID+10490-SOID+966-LOCL+1,00.html
4. Ard-Jan Moerdijk, Lucas Klostermann. Opening the Networkswith Parlay/OSA: Standards and Aspects Behind the APIs. IEEE Network. May/June 2003.
5. Stephen M. Mueller. APIs and Protocols for Convergent Network Services. McGraw Hill. USA 2002.
6. Object Manager Group. http://www.omg.org
7. The Open API Solutions Application Test Suite http://www.openapisolutions.com/applicationtestsuite.html
8. J. Polo, J. Delgado. An easy way to develop mobile and wireless applications. 7th Int. Conf. Mobile and Wireless Communications Networks. To be published (Sep. 2005).
9. The 3rd Generation Partnership Project (3GPP) http://www.3gpp.org/
10. E. Vanem et al. Managing heterogeneous services and devices with the device unifying service. Implemented with Parlay APIs. 8th Int. Sym. Integrated Networks Manag., 2003.
11. E. Vanem et al. Realising Service Portability with the Device Unifying Service using Parlay API. International Conference on Communications, 2003.
12. J. Zuidweg. Next generation intelligent networks. Boston [etc.]: Artech House, cop. 2002.