

Adaptable Services and Applications for Networks

Josep Polo, 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 networks. An API (Application Programming Interface) has been introduced that permits this objective: the Parlay/OSA API. Telecommunication operators offer tools to facilitate the development of those applications and services to use their networks, 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.

1 Introduction

The use of network services is growing. It is adequate that the access to network is opened to facilitate their development. This objective can be achieved using an open API for network service applications that shield most of the detail and eliminates most effort if the development is directly over underlying networks. An API that offers many advantages is Parlay and Open Service Access (OSA).

Although this API facilitates the services development, it is still very complex, difficult and hard to use. To try to solve this situation, many operators and service providers offer their different development tools, SDKs (Software Development Kit), that facilitate the development and implementation phases. These tools simplify the use of the API.

This document begins with an introduction about the Parlay/OSA (P/OSA) API, it shows a general idea about the API and its complexity. Then, it demonstrates how an adaptable service or application 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 or application 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 [13] (see Fig. 1):

- 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.

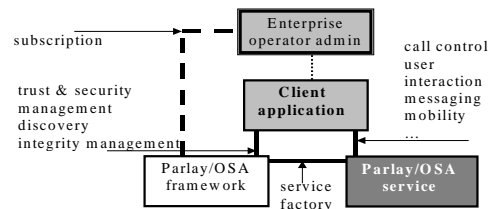


Fig. 1. Parlay/OSA main entities and their relationship.

P/OSA has several interfaces to allow access to different network functionalities or services [13]:

- Framework: security, integrity and management framework.
- Call control: setting up, releasing and managing calls, conferences and multimedia connections; notifications of call and connection related events.
- Data session control: setting up, releasing and managing data sessions.
- User interaction: interaction with users to play or display messages and retrieve user input.
- Mobility: notifications of user location and user status.
- Generic messaging: sending and receiving messages (e-mails, voice mails, SMS), manipulating mailboxes and mail directories.
- Terminal capabilities: to interrogate a terminal to obtain its capabilities.
- Connectivity management: negotiation and management of QoS and service level agreements.
- Account management: create, delete and modify subscriber accounts.
- Charging: reservation and charging of units of volume or money against a subscriber account.
- Policy management: creation and management of policy classes and their parameters. It can define service level agreements.
- Presence and availability management: to allow subscribers and terminals in the network to exchange information about presence and availability.

2.2 Application architecture

When developing a service or application 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 with regard to the services use.
- **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.

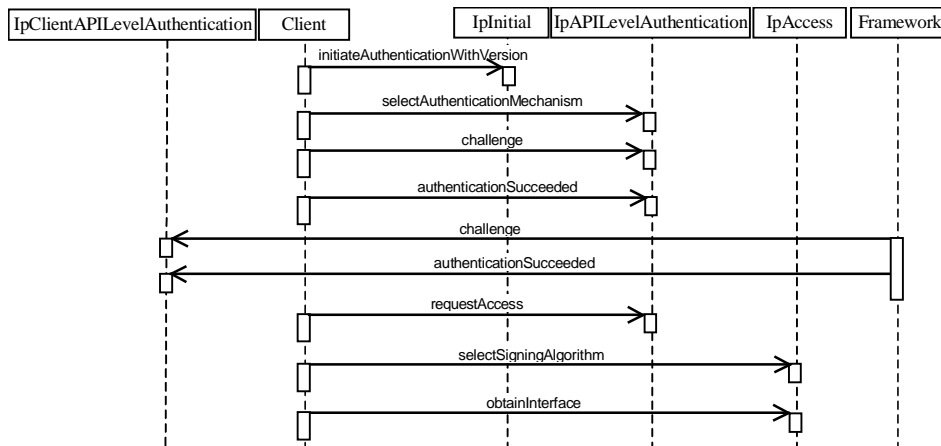


Fig. 2. Initial access of an application to the framework [2].

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, they are shown in Fig. 2. 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 new applications. Usually, these tools are composed by a SDK and a simulator.

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

- Java software libraries. P/OSA defines a language-independent API (applications can be implemented in any language) and they use the Object Management Group (OMG) [8] Unified Modelling Language (UML) and Interface Definition Language (IDL) [7] 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, but 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, testing and demonstration.
- They have a similar structure. They offer a Java library, which simplifies the development of the application, but the application can access directly to the network using 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.** In this step 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.

In order to show how to implement a service or application using these tools, we explain the main steps necessary 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 all 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
- The following sections describe 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 to use 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. For instance, if we selected the User Location service, the sentence to use is:

```
userLocationObject = serviceAdapter.requestLocation(user);
```

It can obtain the geographical location of a single user and waits for the result. The `user` is a string containing a proper value.

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.

As the previous section, we describe the three main steps: authentication, service selection and service use. And finally, the resources release.

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 parameter 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();
```

Finally, the method to use is `dispose` of the class `FWproxy`.

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

This method releases all resources occupied by this instance.

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 differ-

ent 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.

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 consist 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. Fig. 3 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 them 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
}
```

Fig. 3. 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. Fig. 4 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
    }
}

```

Fig. 4. ISG framework class.

As we can see, all general methods (abstract methods in framework class, Fig. 3) 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 Fig. 4. 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
    }
}

```

Fig. 5. 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. Fig. 5 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. Fig. 6 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
}

```

Fig. 6. 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 (Fig. 3), but when the application runs, the real methods used are those in the ISG framework class (Fig. 4) if the MI SDK is used or the NRG framework class (Fig. 5) if the NE SDK is used.

6 Conclusions

Before the P/OSA API was developed, no one, except the network operator, had access to many network resources. Nobody than the network operator can deploy new services or modify them. 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 if the development is directly over underlying networks, those interfaces are very complex, difficult and hard to use. Several SDKs exist that solve these obstacles. The SDKs simplify the use of the P/OSA interfaces, furthermore, 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. This fact presents a great advantage during development or evaluation phase. They can be done in a stand-alone way, without the difficulty that can add the fact to access a real network, and an easy application development, testing and demonstration.

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. These applications are very similar, they are only different when accessing to P/OSA functionality. This means that it is necessary to develop an application for each SDK intended to use. To isolate the developing 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 harder to others. Due to this reason, it is possible that we could not apply this interface to all SDKs. We are currently working on these problems and new papers will cover our future results.

7 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, <http://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. J.W. Hellenthal et al. Validation of the Parlay API through prototyping. Intelligent Network Workshop, 2001.
4. Zygmunt Lozinski. Parlay/OSA – a new Way to Create Wireless Services. IEC Mobile Wireless Data.
5. MiLife™ ISG SDK 3.0 Documentation.
<http://www.lucent.com/products/solution/0,,CTID+2019-STID+10490-SOID+966-LOCL+1,00.html>
6. Ard-Jan Moerdijk, Lucas Klostermann. Opening the Network with Parlay/OSA: Standards and Aspects Behind the APIs. IEEE Network. May/June 2003.
7. Stephen M. Mueller. APIs and Protocols for Convergent Network Services. McGraw• Hill. USA 2002.
8. Object Manager Group. <http://www.omg.org>
9. The Open API Solutions Application Test Suite
<http://www.openapisolutions.com/applicationtestsuite.html>
10. The 3rd Generation Partnership Project (3GPP) <http://www.3gpp.org/>
11. Erik Vanem et al. Managing heterogeneous services and devices with the device unifying service. Implemented with Parlay APIs. IFIP/IEEE 8th International Symposium on Integrated Networks Management, 2003.
12. Erik Vanem et al. Realising Service Portability with the Device Unifying Service using Parlay API. International Conference on Communications, 2003.
13. Johan Zuidweg. Next generation intelligent networks. Boston [etc.]: Artech House, cop. 2002.
14. Alcatel 8601 PGW, Parlay/OSA Gateway. www.alcatel.com
15. Appium – Gbox. www.appium.com
16. Open API Solutions- ATS. www.openapisolutions.com
17. Aepona PCP/SDK. www.aepona.com
18. Kabira Technologies. OSA Gateway Services. www.kabira.com
19. Net4Call Access. www.net4call.com