Evaluation of OSA API's for 3rd party developers of Telco-services

Masters Thesis
in
Information and Communication Technology

Submitted to
Agder University College
Faculty of Engineering and Science

By

Anders Henriksen & Kjetil Ribe

Grimstad, May 2002

Abstract

Changes in business models introduce new actors to the telecom business. A set of new demands occurs and to handle this, a technical enabler like OSA is needed.

Telenor R&D, which often provides Telco services in research projects, must form an opinion concerning this framework. Thus an evaluation of OSA is inquired.

This thesis presents an introduction and evaluations of the OSA framework, further, a closer look at two of the OSA API's (User Status and Terminal Capabilities) are made. To enhance the evaluation, Youngster has been utilized as an example of where the OSA framework can be used. An implementation of one of the API's has also been done with the use of Web Services and CORBA.

Our main conclusion is that OSA has not managed to make the framework user-friendly. Too many details must be implemented to get access to Telco services, and 3rd party developers need to know unnecessary much about telecommunication.

Preface

This thesis is performed to complete the Master of Science programme in Information and Communication Technology (ICT) at Agder University College, Faculty of Engineering and Science in Grimstad.

We would like to thank Alf Martin Sollund and Hans-Thomas Nyberg at Telenor R&D for supervising us during this diploma-work. Also our contact at Agder University College, Lars Line, has been supporting us.
A special thank to Mark H. Butler at Hewlett Packard Company for helping us with DELI.

Last, but not least we would like to thank our girlfriend, Jannicke Knutsen, for being a loving and caring person regardless of our mood swings and long working hours during the completion of this thesis.

Kjetil Ribe

Anders Henriksen

_____

_____

Grimstad, May 2002

Abbreviations

3GPP	3rd Generation Partnership Project
AAA	Authentication, Authorisation and Accounting
AAD	Authentication, Authorisation and Discovery
ADO	ActiveX Data Objects
API	Application Programming Interface
ASP	Active Server Pages
BES	Borland Enterprise Server
CAMEL	Customized Application for Mobile network Enhanced Logic
CAP	CAMEL Application Part
CCM	CORBA Component Model
CC/PP	Composite Capability / Preference Profiles
CORBA	Common Object Request Broker Architecture
CSE	CAMEL Service environment
DELI	Delivery context LIbrary for CC/PP and UAProf
DIFF	Difference
ETSI	European Telecommunications Standards Institute
GIOP	General
GPRS	Global Packed Radio Switched
GUI	General User Interface
HLR	Home Location register
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDL	Interface Description Language
IIOP	Internet Inter-Orb Protocol
IP	Internet Protocol
JAIN	Java API's for Integrated Networks
JDK	Java Development Kit
LBS	Location Based Services
MAP	Mobile Application Part
MD5	Message-Digest Algorithm 5
MExE	Mobile Station (Application) Execution Environment
MIME	Multipurpose Internet Mail Extensions
OMG	Object Management Group
ORB	Object Request Broker
OSA	Open Service Access
PDP	Packed Data Protocol
RDF	Resource Description Framework
RMI	Remote Method Invocation
SAT	Subscriber identity module Application Tool-Kit
SCF	Service Capability Feature
SCS	Service Capability Server
SDK	Software Development Kit
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
TC	Terminal Capability
TCP/IP	Transfer Control Protocol / Internet Protocol
Telco	Telecommunication operators
TTM	Time To Market
UAProf	User Agent Profile
UDDI	Universal Description, Discovery, and Integration
UL	User Location
UML	Unified Model Language
US	User Status

VHE	Virtual Home Envornment
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WGW	WAP Gateway
WGP	WAP Gateway/PushProxy
WML	Wireless Markup Language
WPP	WAP Push Proxy
WSDL	Web Service Description Language
WTA	Wireless Telephony Applications
XHTML	Extensible HyperText Markup Language
XLS	Extensible Stylesheet Language
XML	Extensible Markup Language

Table of Contents

Table of figures

1 Introduction

New players have been introduced to the telecom business due to a change in business models. Some want to address users directly, while others prefer to address users via a network operator. Both way, they compete in the service market, and they need a network. To handle this demand we need a technical enabler; OSA (Open Service Access).

This new business model does not change the fact the network operator "owns" the costumer through the network access.

Today there is not implemented any standardisation of how services, enabled through the telecom operator's networks, are offered by operators to 3rd party service providers. 3GPP has standardised a framework to handle this called OSA.
OSA is using a set of open standardised interfaces to enable operators and 3rd party developers to make use of network functionality.
With the use of standardised API's, and the fact that the technology is open, will lead to several advantages like:

- ✓ Applications will have shorter Time To Marked (TTM).
- ✓ 3rd parties can develop and deploy applications.
- ✓ Network independent applications.

This will inevitably lead to new and innovative services.

In cooperation with Telenor R&D external conditions for our thesis were set. Since Telenor R&D participates in the Youngster project as a Telco provider, they saw the benefit of using our thesis to gain a broader perspective of OSA. In this thesis we will use the Youngster project as an example of how to use the OSA framework and how they can benefit from the use of it.

Youngster will develop technologies to create a new open active mobile multimedia environment that is accessible from anywhere by a wide range of devices and networks and supports context-aware features (including location-awareness) allowing seamless and highly adaptive delivery of services.
Read more about the IST-Youngster project in Appendix A - Youngster.

1.1 Thesis scope

We will define different criteria, which will be used to evaluate the framework. Also comparing similar frameworks will be done.
In particular we will look into the services made available in the Youngster project. Our concentration will be on the Terminal Capabilities- and the User Status API's. During the evaluation, the matters of both the telco-provider and the 3rd party developers will be considered.

Some implementation work will be done to support the evaluation of the framework. During the implementation-phase we will consider different implementation models, implement the Terminal Capabilities API, and evaluate this API in accordance to the framework.

1.2 Objective

The objective in this thesis is to give Telenor R&D an introduction and evaluation of the OSA Framework.

1.3 Outline

This report is structured in five chapters, which will be described briefly.

Chapter 1 gives a short introduction to the thesis; containing scope, objective and outline.

Chapter 2 gives an introduction to OSA general and looks specific into two API's. For each term, we discuss it and come to a conclusion. We are also looking briefly at alternative frameworks compared to OSA.

In chapter 3, implementation of the OSA API, Terminal Capabilities, and a briefly introduction to the distributed models used in the implementation is described.

Chapter 4 contains an overall discussion based on the former chapters.

The final chapter 5 contains the concluding remarks.

2 OSA Evaluation

In this chapter we will define a set of criteria, give a short introduction to the OSA framework and look further into two of the OSA API's.

2.1 Criteria

In the beginning of the assignment we focused on defining a set of criteria, which we later could use as a foundation for evaluating the OSA framework.
We will not use each criterion literally, but we will have them in mind, and use them as base line for each discussion.
After several discussions and reading different specifications, we concluded with a set of criteria, which we meant where important in an evaluation of a framework.

- ✓ Complexity
- ✓ Does the API offer what it claims to offer?
- ✓ Error handling
- ✓ Language and operation system dependency?
- ✓ How is the specification written? (i.e. IDL)
- ✓ Telco developers VS. 3rd party developers
- ✓ Dependencies of other specifications

2.2 Generally

2.2.1 Background

In the end of 1998 and the beginning of 1999, 3GPP formed a group called OSA (Open Service Architecture, later known as Open Service Access). The focus of this group was to define architecture to support the virtual home environment (VHE). This group have made a specification of an API-set for Open Service Access in a release called *Release 99*. This release was defined by both 3GPP and ETSI in cooperation with Parlay. In the next release Parlay was working together with ETSI and 3GPP with the specification, now also cooperating with JAIN.

2.2.2 Introduction

The specification contains information about API's for use by 3rd party developers, mapping to lower layer protocols and a framework.

The concept of the virtual home environment is that the user has access and can see his personalised features in any kind of network, user terminal or wherever the user is located.

"*The OSA defines an architecture that enables operator and 3rd party applications to make use of network functionality through an open standardised API (the OSA API)*."
This is the idea of OSA. The API-set gives applications an independency of underlying network technology.

The expressions *Service Capability Server* (SCS) and *Service Capability Feature* (SCF) are frequently used in the OSA specifications and in 3GPP. It is usefully to get to know these expressions. SCS is the entity that is providing OSA interfaces towards an application while a SCF is the functionality offered by a SCS through OSA interfaces. In a way SCS acts as gateways between the network entities and the applications.

Applications are connected to the Service Capability Servers (SCS's) through the OSA API. The SCS's map these API's to underlying protocols (e.g. Mobile Application Protocol (MAP) and CAMEL application part (CAP)), and hides the network complexities from the application.

OSA defines different network functionality that is offered to applications as a set of SCF's. This functionality is described using OMG's Interface Description Language (IDL). The specification also contains sequence diagrams and state chart using UML as description language.

The main objective of OSA is to provide an extendible and scalable architecture. It should also be open and accessible to 3rd party developers. By using IDL, OSA becomes independent of programming language and operating systems.

OSA consists of two main parts; The Framework and the SCS. In Figure 1 – Overview of OSA we can see these main parts. OSA consists of two set of API; internal and external. The external API is between the application and SCS/Framework, and the internal between the framework and the SCS.

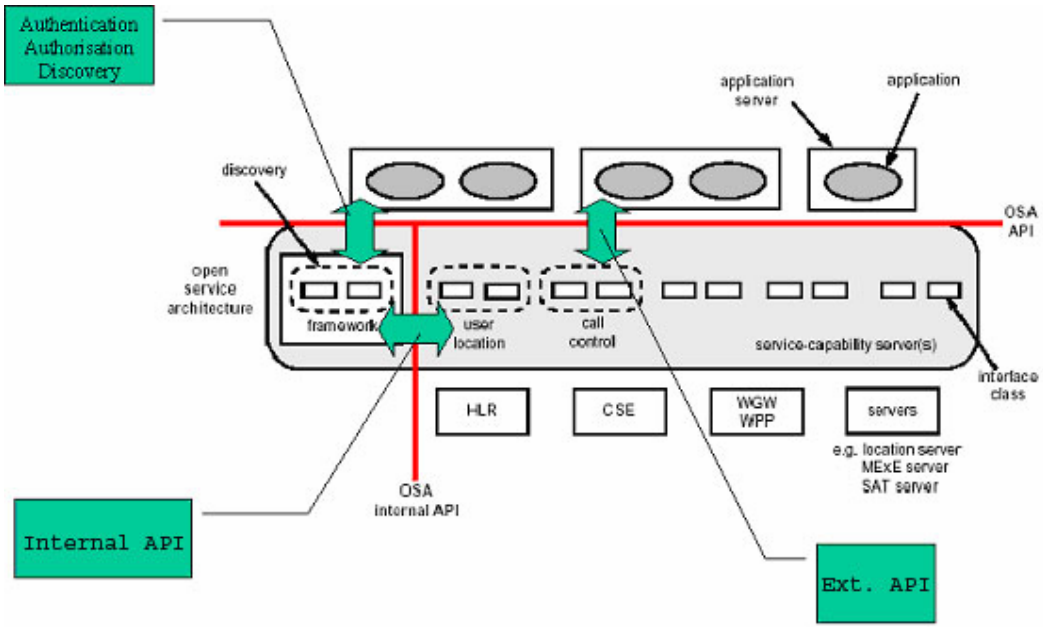

Figure 1 – Overview of OSA

2.2.3 Framework

This part of OSA provides applications access to SCS. Services that the framework provides are authentication, authorisation and discovery. Applications have to get authenticated to get access to the SCS. After this the framework will find which rights this specific application has. This is done through authorisation. After this phase, the framework will tell the application which SCF's this OSA implementation provides. This is called discovery. Every new SCS must register their individual features with the framework.

2.2.4 Service Capability Servers

These servers will provide the applications with network functionalities through Service Capability Features. Example of this can be *User Status* and *Call Control*.

One SCF will provide a number of interfaces and methods that are specified in OSA. We can divide the interfaces into two groups; *internal interfaces* and *external interfaces*.

OSA specifies different API's. These are:

Generic call control (Part 4) [6]

The generic call control SCF provides the basic call control capabilities for the API. It allows calls to be instantiated from the network and routed through the network. The call model is based around a central call model that has zero to two call legs that are active (i.e. being routed or connected), each of which represents the logical relationship between the call and an address. However, the application does not have direct access to the call legs. Generic call control supports functionality to allow call routing and call management for CAMEL Phase 3 and earlier services. Generic call control is represented by the IpCallManager and IpCall interfaces that interface to services provided by the network. Some methods are asynchronous, in that they do not lock a thread into waiting while a transaction performs. In this way, the client machine can handle many more calls than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppCallManager and IpAppCall.

Generic user interaction (Part 5) [7]

The generic user interaction interface SCF's are used by applications to interact with end users. The IpUIManager, IpUI and IpUICall interfaces that interface to service capabilities provided by the network represent the GUI.
The IpUI interface provides functions to send information to, or gather information from, the user, i.e. this interface allows applications to send SMS and USSD messages. An application can use this interface independently of other SCF's. The IpUICall interface provides functions to send information to, or gather information from, the user (or call party) attached to a call. To handle responses and reports, the developer must implement IpAppUIManager, IpAppUI and IpAppUICall interfaces to provide the callback mechanism.

Mobility (Part 6) [8]

The mobility SCF consists of two main parts:

- ✓ Network user location

The network user location (UL) provides the IpUserLocationCamel interface, which provides methods for periodic and triggered location reporting. Most methods are asynchronous, in that they do not lock a thread into waiting while a transaction performs.
In this way, the client machine can handle many more calls than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppUserLocationCamel interface to provide the callback mechanism.

- ✓ User status

The user status (US) basically provides the application with a means of finding out what the terminal is up to at any particular time. This means that it is able to find out when a user completes a call and when the user switches the telephone on, off or busy.

Terminal capabilities (Part 7) [9]

The terminal capabilities SCF enable the application to retrieve the terminal capabilities of the specified terminal. This information is useful for the developer in inter alia front-end considerations. This Service Capability Feature (SCF) provides an interface that is called IpTerminalCapabilities. This interface only contains one synchronous method, getTerminalCapabilities.

Data session control (Part 8) [10]

The data session control provides a means to control, on a per-data-session basis, the establishment of a new data session. This means, especially in the GPRS context, that the establishment of a PDP session (not the attach/detach mode) is modeled. Change of terminal location is assumed to be managed by the lower tier network and is therefore not part of the model. The underlying assumption is that a terminal initiates a data session and the application can reject the request for data session establishment, can continue the establishment, or can continue and change the destination as requested by the terminal. The modeling is similar to the generic call control, assuming a simpler underlying state model. An IpDataSessionManager and IpData Session object are the interfaces used by the application, whereas the IpAppDataSessionManager and the IpAppData Session interfaces are implemented by the application.

2.2.5 Discussion OSA general

The specification of the *framework* makes a god standard of authentication and authorization. The use of UML and IDL gives a full explanation of how the specification should be implemented, and the implementer can choose any kind of implementation language and software platform. It is clearly explained that the SCS's can be implemented on different entities; this can make the OSA a fully distributed system even inside one Telco provider.
With the usage of IDL, structure of the *framework*, and the distributed way of thinking, OSA can easily be compared with the structure of CORBA Component Model.

Since the specification is so fine grained, the threshold of the knowledge of OSA, required by both the implementer and the 3rd party developer, is at a high level. Thus few 3rd party developers outside of the telecom industry have made services/applications using OSA API's. This high demand of knowledge of OSA is based on our experience with different specifications. Several issues like specifications of data-types, error-handling, authentication and authorization give the specification the level of fine grain we mean that the OSA specification has. The thoroughness is also a reason why the specification work has been slower than expected.

Some providers may have given easy access to some of their services, but this has been depending on each Telco provider. The main purpose of OSA is to give 3rd party developers easy access to all Telco services in a standardized way. By doing this, each developer can develop applications without thinking of how difficult telecommunication-matters is implemented and by whom.

Our thesis has focus on the connection between OSA and a 3rd party developer, but in a discussion of OSA it is important not to forget the mapping between OSA and lower tier protocols. This makes a big difference between regular API-set and OSA.

As documented in 2.2.4, we have seen how compound the OSA specification is. This made us think of other solutions. One idea was to define a new layer on top of the OSA API-set to make it easier for 3rd party developers to use the OSA API's. This could be done with Web Services. This extra layer could handle a lot of the "handshaking", and offer 3rd party developers less complex access to Telco services.

In February 2002 we saw some minutes from the OSA/Parlay meeting in Hong Kong. They had discussed the complexity of OSA and the reason why so few 3rd party developers had implemented services based on OSA. On this meeting OSA/Parlay launched the idea of Parlay X. Parlay X is an XML specification of OSA, and is meant to work as a layer on top of the OSA API. This would make it easier for 3rd party developers to implement services using OSA API's. In chapter 4 Discussion, Parlay X versus our solution is further discussed.

2.2.6 General OSA conclusion

The OSA is thorough specified. This makes the API-set well founded, in some cases very fine grained. Services provided by Telco providers have so far been difficult to use for 3rd party developers because of the lack of standardization.
The 3rd party developer still has to know unnecessary much about telecommunication. It seems that the Parlay X is going to make this access easier, but until the release of this specification (planed finished 4 quarter 2002), OSA is too complicated.

2.3 Specific against the IST-Youngster project

2.3.1 Discussion OSA in Youngster

There are several issues in the Youngster project that is dependent on a Telco provider. Example of this is *User Status, User Location, Presence, and Terminal Capabilities*. Some of these are already solved in the project, like *User Location*, without using OSA API's. We will not take this into consideration in our discussion.

Since OSA provide both 3rd party API's and mapping to lower tier protocols, OSA should be an alternative solution for Youngster. In the documentation about Telco services made so far in the Youngster project it has been used the OSA vocabulary, but it has not been looked into each API, and the interfaces that are specified, has not been used. This is caused by not having available OSA implementations in live networks.

Since large specification organs like 3GPP, ETSI and Parlay specifies OSA, OSA is a good attempt from the Telco side on how to provide Telco services. Companies like Ericsson, Alcatel etc. have already made OSA gateways that follow the OSA specification. These gateways provide both the interfaces between Telco and 3rd party developers and the mapping to lower layer protocols.

Because OSA is so compound making a whole implementation of OSA is too demanding on resources for a project like Youngster. The fact that Youngster is at the end of the project makes it more difficult to implement OSA, but it seems important to deliver the API set to 3rd party developers. Thus Youngster can easy meet the future with the use of OSA. The use of a gateway may be a solution, especially if Telenor R&D has one available. This will make the implementation of some of the services that Youngster needs from a Telco provider easier.

2.3.2 Conclusion OSA in Youngster

As we have discussed in 2.3.1 Discussion OSA in Youngster, and the fact that Youngster is a research project, OSA should be considered. Youngster will benefit from using OSA. Youngster can, with the use of OSA, meet the future better prepared. With this, we do not say that Youngster should implement the entire specification, but either use an OSA gateway, or the API-set provided in the OSA specification.
One reason to only implement the API-set is to prevent 3rd party developers from redesign their applications when an OSA gateway is used. Another reason is that OSA is in an early phase, and not all the features are implemented in the gateways.

2.4 Alternative frameworks compared to OSA

To evaluate a framework it is important to look at other alternative frameworks. Since OSA is a Telco specific framework we looked for other Telco framework. This was not an easy job. There is no concrete alternative in the Telco industry to the whole OSA. You can find API-set that provides specific Telco services, but not any that provides a framework like the one found in the OSA specification.

We had to look at other alternatives outside the Telco industry. This meant that specific Telco API's would not be present, but a good framework can be adapted into any systems. The alternative had to be a distributed system, since the Telco provider and the 3rd party developers often are located at different places. Also each Telco service can be located at different entities.

One solution that's quite similar to the way OSA is built is called CORBA Component Model. CCM uses IDL as specification language, has the ideas of one "home" that handles the AAA, is distributed and is well known in the market. This is also a thorough specification, like OSA, which require that the developers and designer have good knowledge about the framework.
It requires a long design phase to use CCM, to get it Telco specific, and it would be different API's from each Telco provider. With this in mind CCM is not a good alternative to OSA.

One other solution is to use an easy standard like Web Services. This standard is easy to implement and supports the idea of a distributed system. Web Services will not replace OSA, but it could be used to provide specific services as discussed in chapter 4 Discussion

2.5 Evaluation of the Terminal Capability API

This API gives the 3rd party developer information on users' terminals. This information is useful for the developer in inter alia front-end considerations.

2.5.1 Overview

The TC API is part 7 [9] of the OSA API standardisation. This Service Capability Feature (SCF) provides an interface that is called IpTerminalCapabilities. This interface only contains one synchronous method, getTerminalCapabilities.

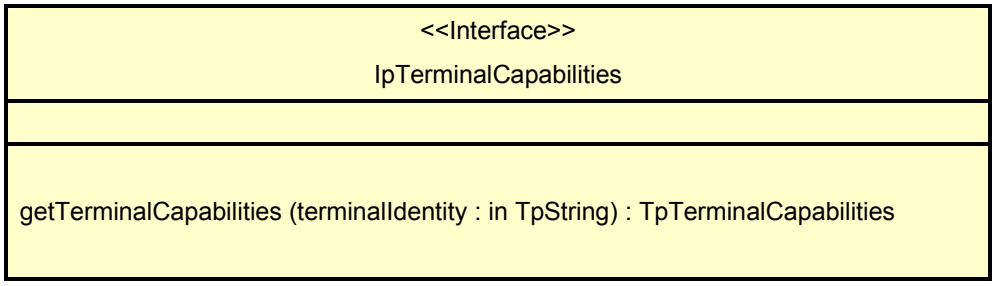

Figure 2 – IpTerminalCapabilities

As we see in Figure 2 – IpTerminalCapabilities, the function getTerminalCapabilities has one in parameter called terminalIdentity. This parameter identifies the terminal, it may be a logical address known by the WAP Gateway/PushProxy.

Name	Type	Documentation
terminalIdentity	TpString	Identifies the terminal. It may be a logical address known by the WAP Gateway/PushProxy.

Figure 3 – terminalIdentity

The return parameter of the function is TpTerminalCapabilities. This parameter specifies the latest available capabilities of the user's terminal. This data type is a Sequence_of_Data_Elements that describes the terminal capabilities. It is a structured type that consists of StatusCode and TerminalCapabilities, as shown in Figure 4 – TpTerminalCapabilities.

Sequence Element Name	Sequence Element Type	Documentation
StatusCode	TpBoolean	Indicates whether or not the terminalCapabilities are available.
TerminalCapabilities	TpString	Specifies the latest available capabilities of the user's terminal. This information, if available, is returned as CC/PP headers as specified in W3C [4] and adopted in the WAP UAProf specification [5]. It contains URLs; terminal attributes and values, in RDF format; or a combination of both.

Figure 4 – TpTerminalCapabilities

The first sequence element tells the application whether the terminalCapabilities is available or not. If it is available the next element is a TpString that is called TerminalCapabilities. This TpString is like CC/PP headers [11]. The CC/PP header is specified in W3C and is returned in Resource Description Framework (RDF) format [25]. In Figure 5 – CC/PP header we can se an example of how the return TpString can look like.

```
<?xml version="1.0"?>
<!-- Checked by SiRPAC 1.16, 18-Jan-2001 -->
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#">

  <rdf:Description rdf:about="MyProfile">

    <ccpp:component>
      <rdf:Description rdf:about="TerminalHardware">
        <rdf:type rdf:resource="HardwarePlatform" />
        <display>320x200</display>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description rdf:about="TerminalSoftware">
        <rdf:type rdf:resource="SoftwarePlatform" />
        <name>EPOC</name>
        <version>2.0</version>
        <vendor>Symbian</vendor>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description rdf:about="TerminalBrowser">
        <rdf:type rdf:resource="BrowserUA" />
        <name>Mozilla</name>
        <version>5.0</version>
        <vendor>Symbian</vendor>
        <htmlVersionsSupported>
          <rdf:Bag>
            <rdf:li>3.0</rdf:li>
            <rdf:li>4.0</rdf:li>
          </rdf:Bag>
        </htmlVersionsSupported>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>
</rdf:RDF>
```

Figure 5 – CC/PP header

The W3C specification says that every attribute in the RDF file is optional. But it says that the UAProf schema MUST consist of one or more CC/PP components, each describing a set of properties (or attributes) within one or more RDF description elements.

W3C specifies that the schema for WAP User Agent Profiles consists of description blocks for the following key components [12]:

- ✓ HardwarePlatform**:** A collection of properties that adequately describe the hardware characteristics of the terminal device. This includes the type of device, model number, display size, input and output methods, etc.
- ✓ SoftwarePlatform**:** A collection of attributes associated with the operating environment of the device. Attributes provide information on the operating system software, video and audio encoders supported by the device, and user's preference on language.
- ✓ BrowserUA**:** A set of attributes to describe the HTML browser application
- ✓ NetworkCharacteristics**:** Information about the network-related infrastructure and environment such as bearer information. *These attributes can influence the resulting content, due to the variation in capabilities and characteristics of various network infrastructures in terms of bandwidth and device accessibility.*
- ✓ WapCharacteristics**:** A set of attributes pertaining to WAP capabilities supported on the device. This includes details on the capabilities and characteristics related to the WML Browser, WTA, etc.
- ✓ PushCharacteristics**:** A set of attributes pertaining to Push specific capabilities supported by the device. This includes details on supported MIME-types, the maximum size of a push-message shipped to the device, the number of possibly buffered push-messages on the device, etc.

2.5.2 Youngsters demands

In the Youngster project it has been specified certain points about the information needed from the Terminal Capability API:

- ✓ Type of device
- ✓ Screen colours
- ✓ Screen size

This information is needed for their Rendering component that is responsible for applying the final transformation process to the WAP/XHTML representation of a screen before it is sent to the user. Rendering will use the Terminal Capabilities to select the appropriate XSL template.

2.5.3 Discussion of the Terminal Capability API

Terminal Capability consists of one interface, and one asynchronous method. The Interface is simple and concrete, but its dependences of WAP seem to be the negative side of the API. The dependency consists of the return-parameter from the function specified in the interface. This parameter is the CC/PP header on RDF format. The header is returned from the WGP, and is an extension in the HTTP header. This will be received from a user in a HTTP request. Knowledge of RDF and CC/PP is required to acquire the information about a terminal's capability. Since this API has strong dependencies to WAP, it also depends on how the WGP have implemented the W3C's UAProf.
The information that this API gives, will fulfil the demands set in the Youngster project if the UAProf is available. If not, an alternative to identify TC is to use the User-Agent string from the HTTP- header instead of the CC/PP header.

2.5.4 Conclusion of the Terminal Capability API

Until UAProf is available from most devices, the Terminal Capability API will not provide the information that a 3rd party developer is interested in, but since this API is a part of the specification, and UAProf most likely will be used in the future, the Terminal Capability can be exploited.
As a conclusion we can also say that Youngster should use this API, in order to follow the new standard. In the near future an OSA/Parlay gateway with the Terminal Capability API will be available. If Youngster wants to use this, the 3rd party developers will not se any changes since the API is already in use by Youngster.

2.6 Evaluation of the User Status API

This API give the 3rd party developer status information on fixed, mobile and IP-based telephony users. This information is useful for 3rd party developer so they always know the current status of a user. This may be online, offline or busy.

2.6.1 Overview

The User Status API is a part of part 6, Mobility in the OSA API specification [8]. This API provides two interfaces, as we see in Figure 8 and Figure 9.

The user-status application interface is implemented by the client application developer and is used to handle user status reports.
The application programmer can use the ipUserStatus interface to obtain the status of fixed, mobile and IP-based telephony users.

The following service capability features will be provided:

- ✓ Retrieval of User Status:
  The application will be able to retrieve the status of the user.

- ✓ Notification of User Status Change:
  The application will receive notifications when the user's terminal attaches or detaches:
  - Detach: the user's terminal is switched off or the network initiates detach upon location update failure.
  - Attach: the user's terminal is switched on or there has been a successful location update after a network initiated detach.

The application will be able to start/stop receipt of notifications for each terminal.

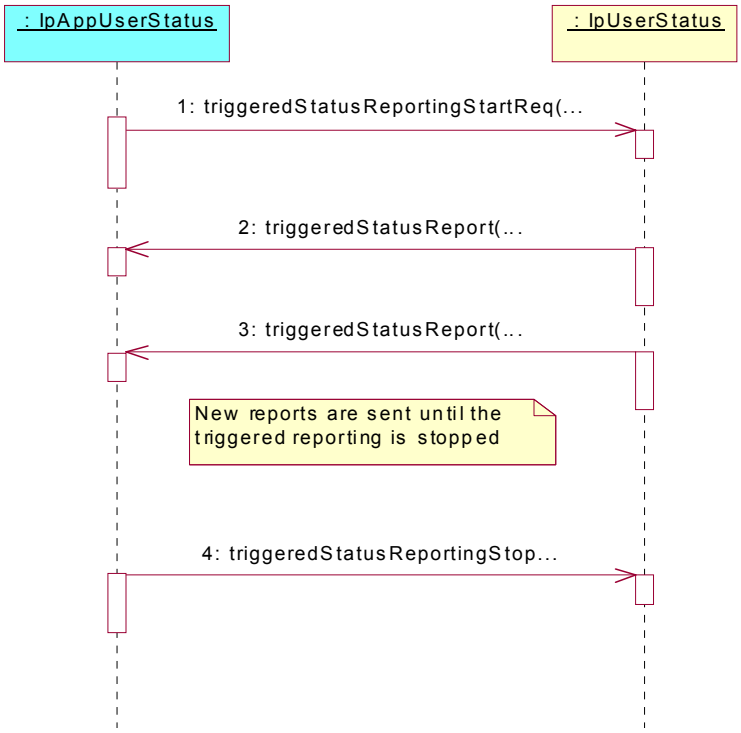

Figure 6 – User Status sequence diagram

Figure 6 – User Status sequence diagram shows how an application requests triggered status reports from the service. It will receive these reports every time a user's status change and the message pass the status to its callback object. This happens until the application sends a triggeredStatusReportingStop.
A callback object is provided by the client-request in order for the server to reply to the client requesting the periodic update.
The service is also able to return errors. If the request contain any errors i.e. parameter error, the service will return an error code.
Also network errors can occur while processing a request from the application. If that happen, the service will call an error method on the application. (*statusReportErr()*)

An interactive request can be made by an application to request a status report from the service, as shown in Figure 7 – Interactive request.

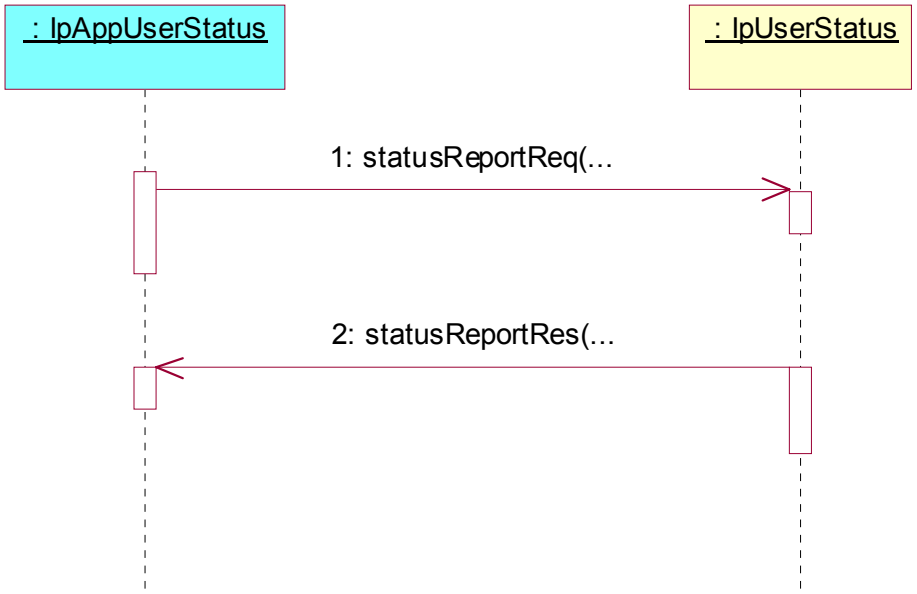

Figure 7 – Interactive request

1: This message is used to request the status of one or several users.
2: This message passes the result of the status request to its callback object.

The IpAppUserStatus interface must be implemented to provide the callback mechanism, and also to handle responses and reports.

<<Interface>> IpAppUserStatus
statusReportRes (assignmentId : in TpSessionID, status : in TpUserStatusSet) : void statusReportErr (assignmentId : in TpSessionID, cause : in TpMobilityError, diagnostic : in TpMobilityDiagnostic) : void triggeredStatusReport (assignmentId : in TpSessionID, status : in TpUserStatus) : void triggeredStatusReportErr (assignmentId : in TpSessionID, cause : in TpMobilityError, diagnostic : in TpMobilityDiagnostic) : void

Figure 8 – Interface Class IpAppUserStatus

The interface shown in Figure 8 – Interface Class IpAppUserStatus, is implemented by the client developer, and it is set to handle user status reports.

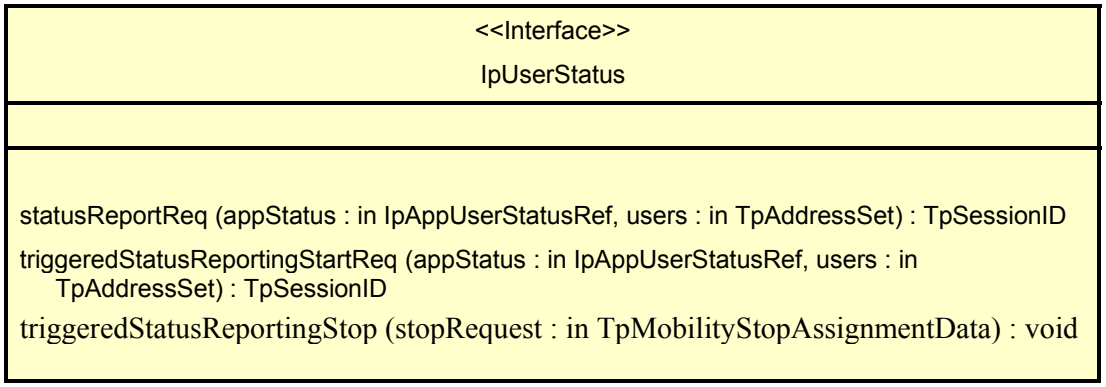

Figure 9 – Interface Class IpUserStatus

The interface in Figure 9 – Interface Class IpUserStatus is used to obtain the status of users. That can be fixed, mobile and IP-based telephony users.

2.6.2 Data definitions

Some of the data definitions in the IpAppUserStatus and IpUserStatus interfaces are shown in the following figures:

- ✓ *TpUserStatus* defines the Sequence of Data Elements that specifies the identity and status of a user.

Sequence Element Name	Sequence Element Type	Description
UserID	TpAddress	The user address.
StatusCode	TpMobilityError	Indicator of error.
Status	TpUserStatusIndicator	The current status of the user.
TerminalType	TpTerminalType	The kind of terminal used by the user.

Figure 10 – TpUserStatus

- ✓ *TpUserStatusSet* defines a Numbered Set of Data Elements of *TpUserStatus*.
- ✓ *TpUserStatusIndicator* defines the status of a user.

Name	Value	Description
P_US_REACHABLE	0	User is reachable
P_US_NOT_REACHABLE	1	User is not reachable
P_US_BUSY (see Note)	2	User is busy (only applicable for interactive user status request, not when triggers are used)
NOTE: Only applicable to mobile (Camel) telephony users.		

Figure 11 – TpUserStatusIndicator

- ✓ *TpTerminalType* - Defines which kind of terminal is used.

Name	Value	Description
P_M_FIXED	0	Fixed terminal.
P_M_MOBILE	1	Mobile terminal.
P_M_IP	2	IP terminal.

Figure 12 – TpTerminalType

- ✓ *TpMobilityStopAssignmentData* -Assignments is used for periodic or triggered reporting of a user's location or status. This data type defines a request to stop the assignment. The request is made up by a sequence of Data Elements listed in Figure 13 – TpMobilityStopAssignmentData.

Sequence Element Name	Sequence Element Type	Description
AssignmentId	TpSessionID	Identity of the session that shall be stopped.
StopScope	TpMobilityStopScope	Specify if only a part of the assignment or if all the assignment shall be stopped.
Users	TpAddressSet	Optional parameter describing which users a stop request is addressing, when only a part of an assignment is to be stopped.

Figure 13 – TpMobilityStopAssignmentData

- ✓ *TpMobilityStopScope* -This enumeration is used in requests to stop mobility reports that are sent from a mobility service to an application.

Name	Value	Description
P_M_ALL_IN_ASSIGNMENT	0	The request concerns all users in an assignment.
P_M_SPECIFIED_USERS	1	The request concerns only the users that are explicitly specified in a list.

Figure 14 – TpMobilityStopScope

- ✓ *TpMobilityError* -Defines an error that is reported by one of the mobility services.

Name	Value	Description	Fatal
P_M_OK	0	No error occurred while processing the request.	N/A
P_M_SYSTEM_FAILURE	1	System failure. The request can not be handled because of a general problem in the mobility service or the underlying network.	Yes
P_M_UNAUTHORIZED_NETWORK	2	Unauthorised network, The requesting network is not authorised to obtain the user's location or status.	No
P_M_UNAUTHORIZED_APPLICATION	3	Unauthorised application. The application is not authorised to obtain the user's location or status.	Yes
P_M_UNKNOWN_SUBSCRIBER	4	Unknown subscriber. The user is unknown, i.e. no such subscription exists.	Yes
P_M_ABSENT_SUBSCRIBER	5	Absent subscriber. The user is currently not reachable.	No
P_M_POSITION_METHOD_FAILURE	6	Position method failure. The mobility service failed to obtain the user's position.	No

Figure 15 – TpMobilityError

- ✓ *TpMobilityDiagnostic* - Defines a diagnostic value that is reported in addition to an error by one of the mobility services.

Name	Value	Description
P_M_NO_INFORMATION	0	No diagnostic information present. Valid for all type of errors.
P_M_APPL_NOT_IN_PRIV_EXCEPT_LST	1	Application not in privacy exception list. Valid for 'Unauthorised Application' error.
P_M_CALL_TO_USER_NOT_SETUP	2	Call to user not set-up. Valid for 'Unauthorised Application' error.
P_M_PRIVACY_OVERRIDE_NOT_APPLIC	3	Privacy override not applicable. Valid for 'Unauthorised Application' error.
P_M_DISALL_BY_LOCAL_REGULAT_REQ	4	Disallowed by local regulatory requirements. Valid for 'Unauthorised Application' error.
P_M_CONGESTION	5	Congestion. Valid for 'Position Method Failure' error.
P_M_INSUFFICIENT_RESOURCES	6	Insufficient resources. Valid for 'Position Method Failure' error.
P_M_INSUFFICIENT_MEAS_DATA	7	Insufficient measurement data. Valid for 'Position Method Failure' error.
P_M_INCONSISTENT_MEAS_DATA	8	Inconsistent measurement data. Valid for 'Position Method Failure' error.
P_M_LOC_PROC_NOT_COMPLETED	9	Location procedure not completed. Valid for 'Position Method Failure' error.
P_M_LOC_PROC_NOT_SUPP_BY_USER	10	Location procedure not supported by user. Valid for 'Position Method Failure' error.
P_M_QOS_NOT_ATTAINABLE	11	Quality of service not attainable. Valid for 'Position Method Failure' error.

Figure 16 – TpMobilityDiagnostic

2.6.3 User Status mapping

The interface class methods defined in 3GPP TS 29.198-6 can be mapped onto CAMEL Application Part (CAP) operations and Mobile Application Part (MAP) operations [24].

2.6.4 Youngsters demands

Demands made by the Youngster project are that the API should deliver the current status of a given end-user. By status, Youngster means whether the user is offline, or online, i.e. connected to GPRS networks or not.
They attend to use this information in inter alia their Context Server. They have a Notifier that checks the Context Server whether the user wants, or is able to be notified.

2.6.5 Discussion of the User Status API

This service provides two types of reporting; you can either request to get a triggered status report. This means that every time a user's status changes, the application will get notified. Or you can make an interactive request. This is the same as the triggered request, but you will only get one response.
Both the trigged status and the requested status gives information if the user is online or offline but the busy status will only be applicable to mobile (CAMEL) telephony users, and it will only be applicable for interactive requests, not when triggers are used. The latter may be a problem for applications that has to know if the user is busy or not. The alternative is to use interactive requests with a periodic timer, or whenever the status information is needed.
An alternative solution drawn by Youngster is to get the user-status from the LBS. This assumes that the user is offline if an error message is returned, and online if not. With this solution, Youngster will not always know if the user is offline or online. Thus, it will not provide the correct status of the user. We assume that this solution will be more time consuming than the trigged request method in User Status. This is because Youngster has to request the user to get the status.

2.6.6 Conclusion of the User Status API

OSA's User Status API provides the 3rd party developer with the information that is required concerning users status. The status of a user is provided in release 4 as [6]:

- ✓ User is reachable
- ✓ User is not reachable
- ✓ User is busy

We will recommend, for Youngster, the use of this API. It is not a solution to implement it, but to use an existing OSA/Parlay gateway. This gateway will provide the API in question.

3 Implementation of Terminal Capability

We decided in cooperation with Telenor R&D to implement the Terminal Capability API. This API's dependency of WAP and Youngsters needs of this API was the reason for this choice.

Youngster needs this API to adapt to devices used by their users to present different content for different devices. So far in the project, Youngster can only differ between WAP users and regular web-browser users, but in the future they want to differ between the devices used due to the increasing number of devices with different characteristics.

Initially we examined UAProf and the WAP specification to find out if it was possible to extract the user profile from a user request. We found different implementations, which made this possible, and we tested one of them called DELI. This implementation is described below.

3.1 DELI

DELI is an open-source library that provides an application interface (API) to allow Java servlets to resolve HTTP requests containing delivery context information from CC/PP or UAProf capable devices and queries the resolved profile.

UAProf header:

```
Format: x-wap-profile = "x-wap-profile" ":" 1#reference
reference = <"> ( absoluteURI | profile-diff-name ) <">
absoluteURI = <a URI as defined by RFC2396>
profile-diff-name = profile-diff-seq "-" profile-diff-digest
profile-diff-seq = ("1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9")
*DIGIT
profile-diff-digest = *OCTET ";" < MD5 message digest encoded by base64 >
DIGIT = <any US-ASCII digit "0".."9">
Default: None
```

Figure 17 – UAProf header

DELI must be implemented on a server capable of processing Java servlets. DELI receives CC/PP or UAProf information from a client-device see Figure 17 – UAProf header, it resolves the received information, and can then pass on this information in order for the appropriate content-provider to adjust the content for that specific user. Profile resolution is central to this process. To identify a terminal UAProf uses profile-diff-digits which is a MD5 message digit encoded by base64. DELI decodes the MD5 message which is a part of the resolving.

One of the goals with CC/PP is that it is supposed to work on even terminals with low bandwidth. In order to do that the client is not sending an entire profile with every request, but only a reference to a profile. The profile is sent only once and then stored on a third device known as a profile repository. The process of interpreting the profile references and differences is known as profile resolution.

This is where DELI comes at hand, as a middle-agent. It performs profile resolution, reassembling packets and then querying profiles in the profile repository.
If DELI is not used, or an equivalent solution is brought at hand; the CC/PP or UAProf information is ignored, thus the server is not able to deliver the content based on the terminal capabilities.

Unlike any HTTP based solution, the CC/PP based solution is always up to date. It doesn't rely on a database or manually updating. DELI just read and interprets the CC/PP profile as a request is made.

When DELI interprets a request made by a terminal that is not providing any CC/PP or UAProf information, it will determine which type of terminal it is in a database providing information for old terminals based on the User-Agent string.

```
Format: x-wap-profile-diff = "x-wap-profile-diff" ":" profile-diff-seq";" profile-desc
profile-diff-seq = ("1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9")
                        *DIGIT
profile-desc =<XML document containing profile subset of the schema defined
in Section 7>
DIGIT = <any US-ASCII digit "0".."9">
Default: None
```

Figure 18 – UAProf DIFF header

Terminals will if a certain characteristic change sends a UAProf DIFF header. This header contains information on with characteristic that has been changed and the new value. If DELI receives a UAProf DIFF header, Figure 18 – UAProf DIFF header, it will update the repository with the new information. This way changes in the profile do not consume large bandwidth, as it would if it sent the whole profile again.

DELI is an alternative to the OSA terminal capability API. It can be used as a stand alone API to resolve the terminals capability. But it can also be used together with the OSA API, as a lower tier for the API on the Telco side. This way the 3rd party developer will only see the OSA API, use the framework and AAD, and get the terminal capability from the OSA TC API.

The input parameter in the function *getTerminalCapability* can be the UAProf header; which in the specification is a string (*tpString)* that identifies the terminal, thus the UAProf header as an input parameter is according to the specification.

During our testing of DELI we had regular contact with Mark H. Butler. He is the leader of the project DELI at Hewlett-Packard Company.

3.2 Design

We decided to implement this API with two different implementation models and languages. Before we decided which models, we started the design phase. In this phase it was important to follow the OSA framework with a Telco provider and a 3rd party developer. Thus the architecture of the system became like this:

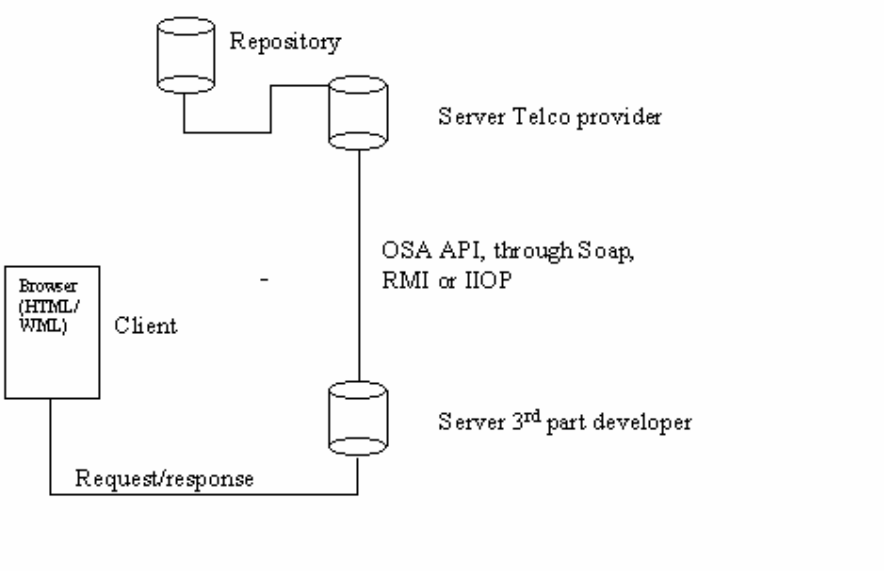

Figure 19 – Architecture

The communication between the Telco provider and a 3rd party developer had to be solved with a distributed model. We had different alternatives like Java RMI, CORBA, and Web Services etc. Since both CORBA and Web services are well known technologies, we decided to use these models.

3.3 Web Services

Web Services is a kind of web-based applications. It encapsulates all necessary information about the service it provide, both what it shall do and what it really does. This way the implementation of the function is not accessible from the outside, and a change in the implementation of the function does not affect the use of the function. A Web Service can be published, located and used across the Internet. It is not depended on a software platform or an implementation language.
Web Services is based on three roles:

- ✓ Service Publisher
  - Provides an infrastructure and maintains a registry of available Web Services. And publishes Web Services to a broker.
- ✓ Service Broker
  - Organize best match between publisher and consumer of a Web Service
- ✓ Service Consumer
  - The one who asks a Service Broker of a given functional service, and uses the one or those Web Services which fits with the demands.

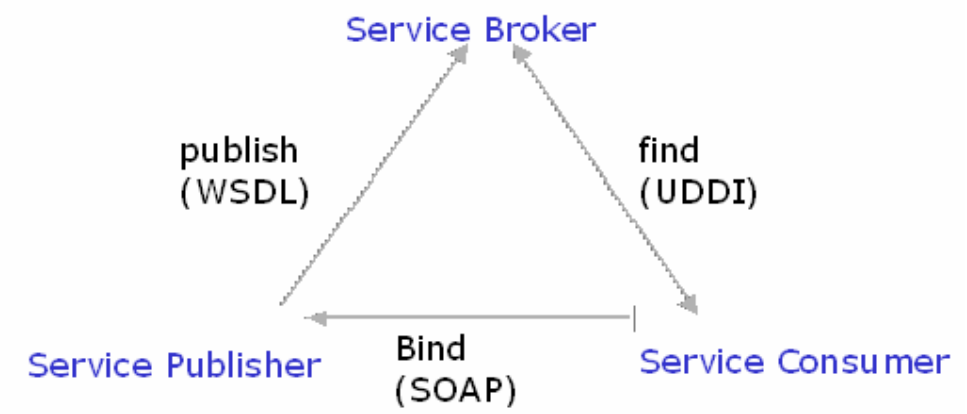

Figure 20 – Web-services three roles

In Figure 20 – Web-services three roles we see how these three roles interact. Web Services contains of four basic elements, the figure shows three (WSDL, SOAP and UDDI) additional to these are XML. SOAP is, according to W3C, a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses [23].

XML is perfect to move structured data over a network. In this way data representation is not bounded to a particular hardware or software demand. These data can be read and manipulated in many different ways. SOAP uses HTTP requests to execute function across a network. A Web Service can interact with remote computers and application. This is done by using HTTP or other transport protocol (e.g. SMTP) as a transport of XML messages.

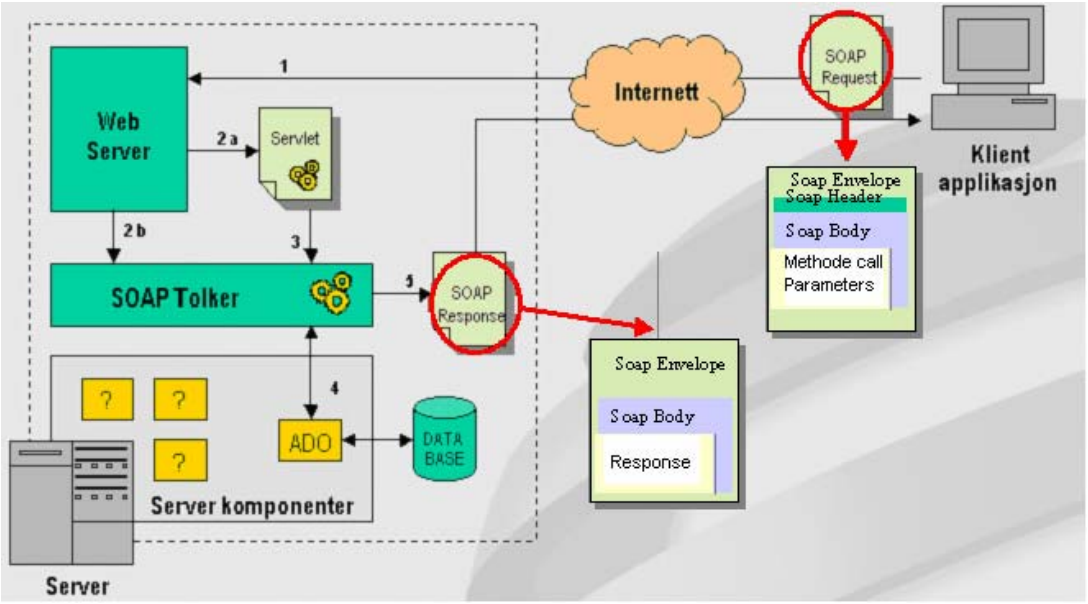

Figure 21 – Web Services architecture

SOAP is an easy protocol that contains of three parts (see Figure 21 – Web Services architecture):

- ✓ SOAP Envelope
  - A *cover* that tells what the message contains of.
- ✓ SOAP Header
  - Sets of coding rules declaring data types etc.
- ✓ SOAP Body
  - A description of methods, parameters and return values.

```
Request:

POST /TerminalCapabilities/Service1.asmx HTTP/1.1
Host: 128.39.203.53
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://microsoft.com/webservices/getTerminalCapabilities"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getTerminalCapabilities xmlns="http://microsoft.com/webservices/">
      <terminalIdentity>string</ terminalIdentity >
    </getTerminalCapabilities>
  </soap:Body>
</soap:Envelope>
```

```
Response:

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getTerminalCapabilitiesResponse xmlns="http://microsoft.com/webservices/">
      <getTerminalCapabilitiesResult>string</getTerminalCapabilitiesResult>
    </getTerminalCapabilitiesResponse>
  </soap:Body>
</soap:Envelope>
```

Figure 22 – SOAP request

In Figure 22 a SOAP request and response is listed. This SOAP request is calling the function getTerminalCapabilities at 128.39.203.53, with a parameter called terminalIdentity which is a string. The response is returned in a tag called getTerrminalCapabilitiesResponse with the getTerminalCapabilitiesReulst as result tags.

UDDI is a catalog service where you can search and find services from different actors. It uses the XML based WSDL to describe services. UDDI can be used internal in a company, branch specific, news, search service, etc.

Web services can be implemented on different platforms and with different implementation languages. Below are listed two different environments.

Java environment
- JDK 1.3
- Apache soap
- Tomcat 3.2.1
- Xerces XML parser 1.2.3
- Windows/Linux

Microsoft .NET environment:
- Microsoft .NET SDK
- Microsoft Internet Information Server
- Windows 2000

Microsoft .NET is a new technology and expected to be adopted by a large number of developers and designers. This is the reason why we have chosen .NET as the environment for our Web Service implementation.

The architecture of our Web Service implementation became like this:

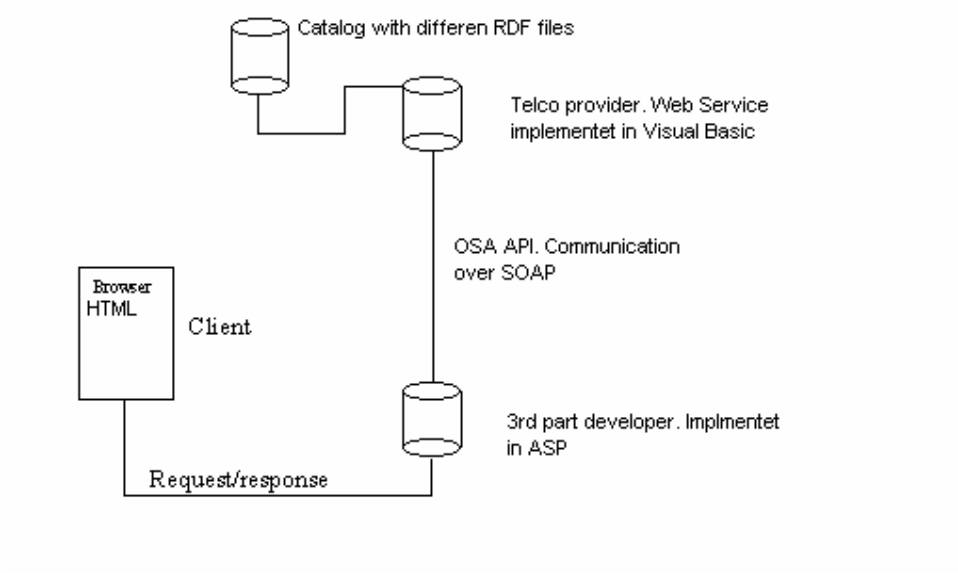

Figure 23 – Implemented Web Service architecture

3.3.1 Experience with Web Services and the Terminal Capability API

The Terminal Capability API is a quite simple API. It got one interface with one method. So this should make it possible to implement. Since Web Services is a new technology, there were several different challenges. Also the fact that we used Microsoft .NET, made a couple of problems with e.g. the access to the Web Service.

Using .NET should be easy, it generate a lot of code for you, maybe too much. Sometimes you loose control of what you are developing. We wrote some of the code in Notepad, instead of using the pre made code in .NET.

When every bug were fixed, the Web Service was easy to access and use. To make a Web Service (Telco side) and to use it (3rd party developer) is not a problem after you have tried a couple of times.

When using just HTTP there is no security. Every function request and response is sent in clear text. For sensitive information this is not a good idea. But by using HTTP secure (HTTPS) each HTTP message are send cryptic, and the security is much better. With the use of Web Services, the problems with access through a firewall are avoided. Web Services is using HTTP as transport protocol, and therefore the communication is through the common web-port (port 80).
We will later come to a conclusion about the Terminal Capability API from OSA seen from a developer point of view. But we can say that the specification gives to little information on how to *get* the terminals capabilities.

3.4 CORBA

CORBA is a standard maintained by the largest software consortium in the world; OMG. CORBA is a middleware design that allows applications programs to communicate with another without regard to their programming languages, hardware/software platforms, the network they communicate over and their implementers.

First you define interfaces in CORBA's IDL, see Figure 24, build application, and client access methods in IDL interfaces of CORBA objects by means of remote method invocation. The middleware that handles the RMI is called ORB.

```
module TerminalCapabilities_server {

  struct TpTerminalCapabilities {
    boolean StatusCode;
    string TerminalCapabilities;
  };

  interface IpTerminalCapabilities {
    TpTerminalCapabilities getTerminalCapabilities(in string terminalIdentity);
  };
};
```

Figure 24 – IDL example

To communicate between ORB's CORBA is using a protocol called GIOP. This is a general protocol that can be implemented on any transport layer with connections. The implementation of GIOP for the Internet uses the TCP/IP protocol and is called IIOP.

When you have written the IDL you use an IDL compiler specific for the language you want to implement the application in. Figure 25 illustrates this [1]:

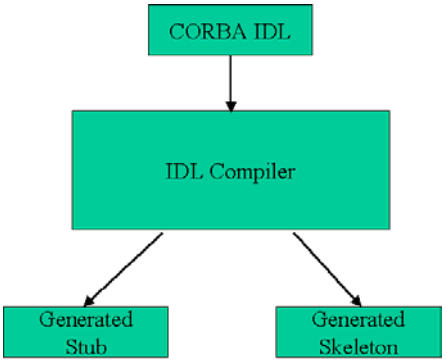

Figure 25 – The IDL Compiler turns IDL into stubs and skeletons

When executing the IDL Compiler there will be generated two files; a *Stub* and a *Skeleton*. The Stub file has to be placed on the client side and the Skeleton file on the server side.

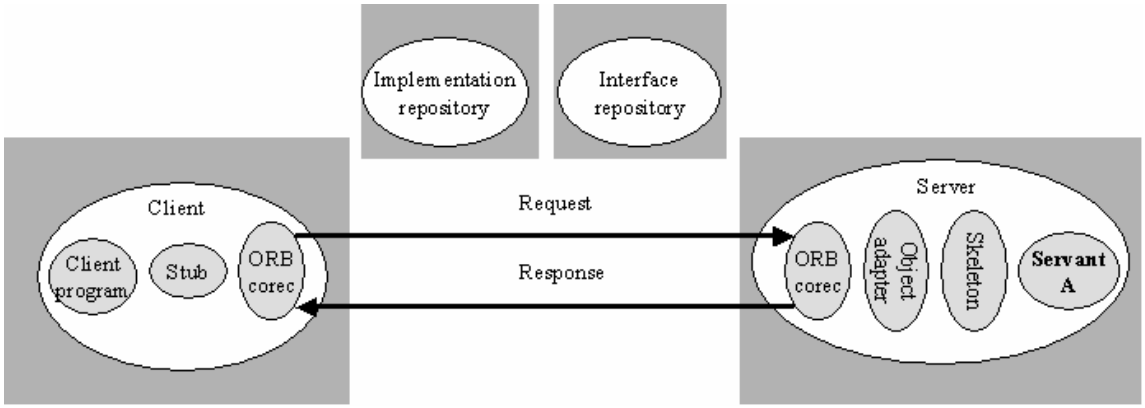

Figure 26 – The main components of the CORBA architecture

In Figure 26 – The main components of the CORBA architecture, we see [3]:

- ✓ ORB core. Carry out the request-replay protocol. The ORB core also provides invocation semantic.
- ✓ Object adapter. The object adapter is a bridge between CORBA objects with IDL interfaces and the programming language interfaces. It creates object references for CORBA objects and activates the objects.
- ✓ Skeleton. RMI are posted via a skeleton to a particular servant, and the skeleton unmarshals the arguments in the request message and marshals the result in the replay message.
- ✓ Stub. The stub marshals the arguments in an invocation request and unmarshals results in replies.
- ✓ Implementations repository. This repository is responsible for activating registered severs on demand and for locating servers that are currently running.
- ✓ Interface repository. The interface repository provides information about registered IDL interfaces to clients and servers that require this information. Not all ORB's provide an interface repository.

Since we chose .NET as the implementation environment for Web Services we decided to use Java for the CORBA implementation. The following environment was used.

Sun JDK 1.2.2
Inprise VisiBroker 4.5.1
Tomcat 4.0
Borland Enterprise Server 4.5
Windows 2000

We used Jbuilder 6.0 as programming application. As we se in Figure 27 – Implemented CORBA architecture the architecture were almost the same as the Web Service architecture.

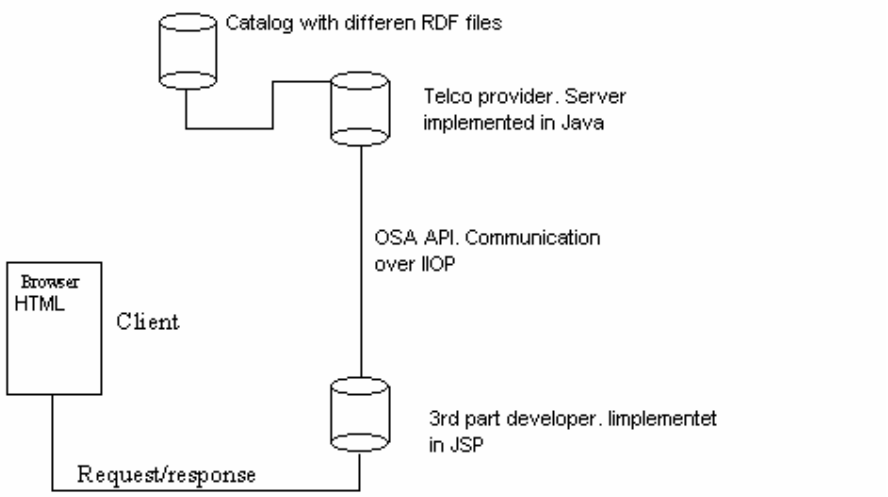

Figure 27 – Implemented CORBA architecture

3.4.1 Experience with CORBA and the Terminal Capability API

CORBA is a more compound model than Web Services. Together with this complexity the use of different JDK's, ORB's and web servers makes a lot of different challenges. We started our development with the use of JDK 1.3.1 and Visibroker 4.5.1, after several tests and reading on the Internet, we came to the conclusion that Visibroker 4.5.1 and JDK 1.3.1 doesn't work together. We then installed JDK 1.2.2 and everything worked well. We then had made a dynamic HTML page that a client could access. This page was made with HTML, JSP, JavaBean and the use of the stub file created by the IDL2Java compiler so we could access the Telco provider function called getTerminalCapability. The client then got returned his user profile.
The benefit of using CORBA instead of Web Services is that you can return an object, which in our case a Sequence_of_Data_Elements called TpTerminalCapabilities. This is not possible in Web Services, where we had to return a string, and use ";" as delimiter between the StatusCode and the TerminalCapabilities.

3.5 Discussion/conclusion implementation

The Terminal Capability API is a simple API. It consists of one interface and one method. This makes the implementation of it, and the use of it, easy. During our implementation we had only technical problems with the different models we used and not any problem with understanding and implementing the API.

To get a communication between the Telco provider and the 3rd party developer was practicable. The problem was to implement the functionality behind the API. In the specification there is no mapping to telecommunication systems, it is only mentioned that it returns a CC/PP header in RDF format. The specification also says that we should use the WAP UAProf-header to get the terminal capability. We tested an existing implementation called DELI. This API could work together with the OSA TC API.
In order to use DELI you need a UAProf capable device. At this date we have only found three such devices; Ericsson T39, Ericsson T68 and Trium Eclipse.
For other non-UAProf capable devices, the old method of requesting the User Agent string in the HTTP header is needed, and then to look up the appropriate information in a legacy database.

4 Discussion

In each chapter we have discussed the themes that have been brought up. We will now combine these previous discussions and finally, in chapter 5, come with an overall conclusion.

OSA is a one-of-a-kind in the market. There is no other, finer grained specification that is specific for the Telco providers, uses standard descriptions, and API known to people outside the Telecommunication industry. The combination of ETSI, 3GPP, Parlay and JAIN has been strengthening to the OSA specification work. OSA, with the use of IDL and UML as description methods, is prepared for new design and implementation technologies, like component based development. It is independent of software platforms and implementation languages.

OSA is a complete framework. It has specified elements like authentication, authorization and discovery.

In addition to provide API set to 3rd party developers, OSA also specifies mapping to underlying network technologies. This makes the specification attractive to Telco providers.

Some of the API's are more thoroughly prepared than other; the User Status is an example of this. In the User Status specification several state charts and sequence diagrams explains how the API should work. The API provides both requested and trigged methods, and the error handling is thoroughly done.
There is also a specification for handling the mapping to underlying network technologies; like MAP and CAMEL.

Other API's like the Terminal Capability is not so thoroughly prepared. The API is weak about what exactly should be returned from the methods and there is no information on how the Telco provider actually gets the capability from a terminal. This API is dependent of several WAP specifications, and how each terminal has implemented UAProf. Our experience with the Terminal Capability API is that few terminals have implemented UAProf, and therefore getting a correct and updated version of a terminal's capability is difficult. But at this point we can't see other alternatives to the UAProf to get the terminals capability, and in the future more and more terminals will be UAProf capable.
During our implementation of the Terminal Capability API we experienced the dependencies to WAP specifications regarding the UAProf and CC/PP header and the different ways terminal vendors have implemented this. This made the implementation of Terminal Capabilities difficult. The communication between the 3rd party developer and the Telco-provider is specified in a way that makes the implementation obvious.

Early in our project phase we discussed the fine grain of the OSA specification. It seemed that the 3rd party developers had to know too much about telecommunication to make services based on the OSA API's. We thought of a solution with a layer on top of the OSA API at the Telco side. This layer should perform some of the handshaking that is specified in OSA. Coarse grained API's could be provided to the 3rd party developers, e.g. in forms of Web Services. Our idea was to do some of this in the implementation phase, but in February we saw in a report from a Parlay group meeting in Hong Kong, where they had discussed the complexity of OSA, and had come to a conclusion: the complexity is the reason why so few 3rd party developers have implemented services built on the OSA technology. They started a group called Parlay X. This group specified a XML based layer on top of OSA to make it easier for 3rd party developers to access telecommunication services. Since OSA had come to this conclusion, we saw no reason to go any further with our solution other than outlining it in this chapter.

Further in this discussion, we will use Youngster as an example of where OSA can be used.
Since the Youngster project contains of both 3rd party developers and Telco-providers, the features with AAD is unnecessary. Authentication of Youngster-clients will be handled in the Youngster Service Platform and not the Telco-provider. The developers do not want/need to perform the authentication and authorization phase. They know which services the Telco provider offers, and therefore also the discovery phase is unnecessary. What Youngster really need is an easy API-set for the specific services they are going to use with a Telco provider. These are services like User Location, User Status, Presence, Terminal Capabilities, etc. OSA specifies both the framework and the services thoroughly and there are no problems using only the services as an easy access to telecommunication systems.

In Youngster the User Status information is needed in the Context Server. Entries in the Context Server will be made when Youngster needs information on a user's status. By using the trigged method in the OSA User Status API, Youngster will always have the updated information on the User Status. To use OSA's specification, Youngster will both have a specification of how to implement the User Status on the Telco side, and a good specification on the API between Telco and 3rd party developers. In the Youngster documentation there is an alternative solution illustrated. This is to get the user status in form of an *error* or *succeed* response from the LBS. This is a more time consuming solution and the Context Server will not always have the updated user status.

The Terminal Capability API is needed in Youngsters Rendering component so they can apply the final transformation process to the WML/HTML representation of a terminal before it is sent to the user. To give the correct representation Youngster is dependent on having the updated information on a user's UAProf.

5 Conclusion

OSA has not managed to make the access to Telco services for 3rd party developers easy enough. The specification is very fine grained and still too many telecommunication matters have to be implemented by the 3rd party developer.
This fine grain makes OSA unique in the marked; the combination of the API-set specification, framework, and mapping to Telco specific protocols.
Because of this, OSA has to be considered used by Telco-providers, but since the specification's thoroughness has made it difficult for the non-telco industry, few 3rd party developers has made services based on OSA API's. Waiting for the Parlay X specification is a solution.

Based on our evaluation of the User Status API discussed in chapter 2.6.5, this API provides what both 3rd party developers and Telco-providers demands concerning the API, the mapping to telco-protocols, and the information User Status returns.

Referring to our previous discussions in chapter 2.5.3 about Terminal Capabilities, we recommend Youngster to use this API. This for providing Telco services on a standardised way and the fact that UAProf will be implemented on most terminals in the future.
To implement Terminal Capability we recommend Youngster to use CORBA as middleware and DELI to get the terminals capability on the server side.

With the use of OSA in Youngster, and other similar projects, an API set to 3rd party developers will be provided in a standardised way. There is no need to implement the entire specification, but either to use an OSA gateway, or the best solution; implement the API's that Youngster needs from the OSA specification. Some of the API's are more compound than other and therefore the use of a gateway is a solution.

6 Reference

[1] Java distributed Objects by Bill McCarty and Luke Cassady-Dion

[2] The Youngster project. (http://www.ist-youngster.org)

[3] Distributed systems; Concepts and design by George Coulouris, Jean Dollimore and Tim Kindberg

[4] 3GPP's specification of the OSA API Part 1. 3GPP TS 29.198-1 V4.3.0 (2001-12)

[5] 3GPP's specification of the OSA API Part 3. 3GPP TS 29.198-3 V4.3.0 (2001-12)

[6] 3GPP's specification of the OSA API Part 4. 3GPP TS 29.198-4 V4.3.0 (2001-12)

[7] 3GPP's specification of the OSA API Part 5. 3GPP TS 29.198-5 V4.3.0 (2001-12)

[8] 3GPP's specification of the OSA API Part 6. 3GPP TS 29.198-6 V4.3.0 (2001-12)

[9] 3GPP's specification of the OSA API Part 7. 3GPP TS 29.198-7 V4.3.0 (2001-12)

[10] 3GPP's specification of the OSA API Part 8. 3GPP TS 29.198-8 V4.3.0 (2001-12)

[11] Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation (http://www.w3.org/TR/NOTE-CCPP). W3C note 27 July 1999

[12] WAG UAProf Wireless Application Protocol WAP-248-UAPROF-20011020-a
http://www1.wapforum.org/tech/documents/WAP-248-UAProf-20011020-a.pdf

[13] Delivering Seamless Services in Open Networks Using Intelligent Service Mediation by Michel L. F. Grech, Robert D. McKinney, Sharad Sharma, John J. Stanaway, Jr., Douglas W. Varney, and Kumar V. Vemuri
http://www.itpapers.com/cgi/PSummaryIT.pl?paperid=9202&scid=100

[14] Parlay Technical overview by Julian Richards and Erik van der Velden. A presentation on the Parlay Group Member Meeting 27 – 29 June 2000 (Download:
http://www.parlay.org/docs/june2000mm/05 Parlay Technical Overview J. Richards E. van der Velden.pdf)

[15] Parlay / OSA APIs in new Telco Eco System, by Holger Schreyer.
http://www-dse.doc.ic.ac.uk/Events/opensig-2001/SchreyerInfitel.pdf

[16] The OSA API and other related issues, by R M Stretch.
http://www.bt.com/bttj/vol19no1/stretch/stretch.pdf

[17] DELI and Mark H. Butler
http://www.hpl.hp.co.uk/people/marbut/

[18] Parlay/OSA: an open API for service development, by Chelo Abarca, Andy Bennet, Ard-Jan Moerdijk and Musa Unmehapo.
(Download: http://www.3gpp.org/tb/cn/cn5/n5-020130_allaboutparlayosa.ppt)

[19] UML Components; A simple Process for Specifying Component-Based Software, by John Cheesman and John Daniels.

[20] Microsoft .NET show. (http://msdn.microsoft.com/theshow/Archive.asp)

[21] Web Service learning (http://www.uddicentral.com/)

[22] **WSDL is governed by the World Wide Web Consortium. Full information on the specification of the standard can be found at the W3C site.(http://www.w3.org/TR/wsdl)**

[23] **Simple Object Access Protocol (SOAP) 1.1 (http://www.w3.org/TR/SOAP/)**

[24] **3GPP's specification of mapping for OSA API Part 6. 3GPP TS 29.998-6 V4.3.0 (2001-12)**

[25] **Resource Description Layer (RDF) Model and Syntax Specification. Lassila O. Swick R. W3C Working Draft (http://www.w3.org/tr/WD-rdf-syntax)**

7 Appendix

7.1 Appendix A - Youngster

The Youngster Project is a European collaboration involving different players within mobile telecommunication. Sony and Siemens contribute with knowledge of mobile terminals, T-Systems Nova and Telenor are tele operators, Steria is a French software company, Norwegian Broadcasting Corporation contributes with content expertise, while Heriot-Watt University, Edinburgh, is experienced in technological research.

The aim of the project is to find out how young people want to use mobile services in the future and to produce these services. The partners will develop technologies to create an active mobile multimedia environment that is accessible from anywhere by different devices and networks.

The innovative, open active mobile service platform (MSP) will support a generation of enhanced mobile services, also to be developed within the project. These services will be personalized and support context awareness, including location/positioning information. Any information may be filtered according to the users preferences, and the security and privacy of the users will be ensured at all times. The project has

Through various service modules of the MSP the users will be able to create their own active and personalized services. Thereby having the opportunity to explore, adjust and create their own mobile communication environment and information space.

New business models will be designed for our target group as they cannot afford premium services. They will therefore be examined to ensure that developments arising from this project can be marketed to different user groups. At the end of 2002 the partners will present an open platform for mobile services for young people to the EU Commission which funds the project through the IST programme.

Milestone date	**Anticipated Project Results**
January 2001	Project start
June 2001	Application Scenario; Mobile Service Platform (MSP) Functionality
October 2001	MSP Base System
May 2002	MSP Final System
October 2002	End of User Trials
December 2002	End of the Project

7.1.1 Youngster Partners

Heriot-Watt University,UK
The Computing & Electrical Engineering Department has extensive experience in research and development in software development as well as database and information systems.
www.cee.hw.ac.uk

NRK - Norwegian Broadcasting Co. Ltd., Norway
Among the front-runners when it comes to content on mobile and traditional devices. NRK has several youth targeted newsrooms and programme divisions, and is the project´s only content provider.
www.nrk.no

Siemens AG, Germany
World leading in electrical engineering and electronics. Possesses core competencies within the next-generation Internet, wireless communications and advanced electronics.
www.siemens.com

Sony International (Europe) GmbH, Germany
Entertainment company and leading manufacturer of audio, video, communications and information technology products for the consumer and professional markets.
www.sony-europe.com

Steria, France
Designs and implements IT solutions for the development of companies within the sectors of banking, telecommunications, transport, industry and power production, insurance and services.
www.steria.com

T-Systems Nova GmbH Berkom, Germany
Berkom is the application-oriented innovation center of T-Systems Nova, a fully-owned subsidiary of T-Systems. Its primary task is to conceive, develop, test and roll out new products, applications and services for the telematics sector.
www.t-systems.de

Telenor AS, Norway
The company is Norway´s leading distributor of voice, information, knowledge and entertainment through a broad range of modern communications services.
www.telenor.com

7.2 Appendix B – Task description

The background for this assignment is the desire of the project Youngster to make available services delivered by telco providers to 3rd party developers.
Today there is no standardization of how these services are presented, thus 3GPP has standardized a framework to handle this. The Youngster project need to make a decision of how they are going to present their services, thus an appraisal of 3GPP's OSA is needed.
As shown in Figure 28 – Architecture, the focus in this assignment will be the interfaces between the OSA and the 3rd party developer. We will define different criteria which will be used to evaluate the framework. Also considerations about different similar frameworks will be made.

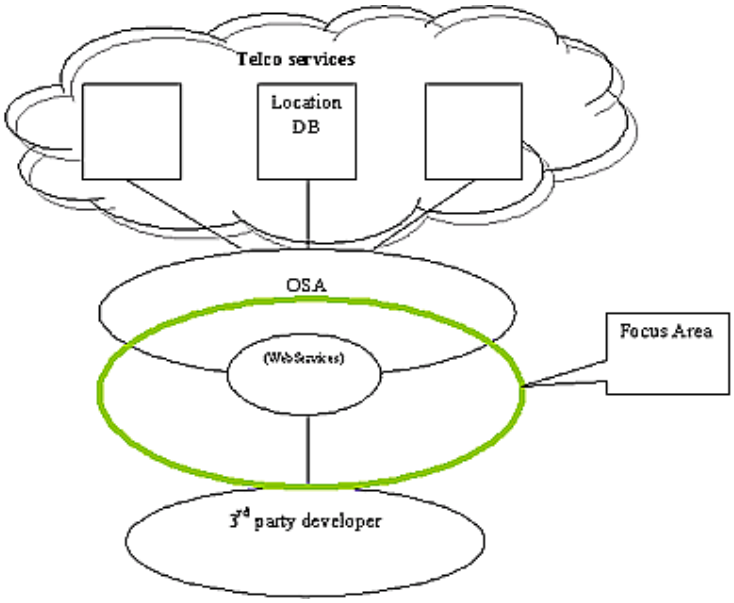

Figure 28 – Architecture

In particular we will look into the services made available in the Youngster project. Our concentration will be on one, or maybe two of the following API's:

- ✓ User Status (US) API - The User Status service capability features enable an application to retrieve the user's status, i.e. to find out on which terminals the user is available.
- ✓ Terminal Capability API - enables the application to retrieve the terminal capabilities of the specified terminal.
- ✓ Presence API - Information about the context of a user, such as disposition and activity status.
- ✓ User Interaction (UI) API - User Interaction service capability feature is used by applications to interact with end users.

- ✓ Call Control (CC) API - The Generic Call Control Service (GCCS) provides the basic call control service for the API. It is based around a third party model, which allows calls to be instantiated from the network and routed through the network. The Multi-party Call Control service enhances the functionality of the Generic Call Control Service with leg management. It also allows for multi-party calls to be established, i.e. up to a service specific number of legs can be connected simultaneously to the same call.

During the evaluation, the matters of both the telco-provider and the 3rd party developers will be considered.
Some implementation work will also most likely be done to support the evaluation of the frameworks through participation in the IST project Youngster. During the implementation-phase we will consider different implementation models, implement a pair, and evaluate them in accordance to the framework.