A Base Solution for Exposing IMS Telecommunication Services to Web 2.0 Enabled Applications

Florian Deinert, Alin Murarasu, Andreas Bachmann, and Thomas Magedanz

Fraunhofer Institute for Open Communication Systems (FOKUS)
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
{florian.deinert,alin.murarasu,andreas.bachmann,
thomas.magedanz}@fokus.fraunhofer.de
http://www.fokus.fraunhofer.de

Abstract. The convergence of telecommunication and Web 2.0 services is leading to new opportunities for the telecommunications market. Companies are looking for ways to include their services in Web 2.0 applications. Predictions suggest that future telecommunication networks will be based on the IP Multimedia Subsystem (IMS), an all IP telecommunication core network. This paper describes an approach to combining Web 2.0 enabled applications, namely widgets, with telecommunication features using IMS. Widgets are small applications based on Web technologies that run on the client device. A new abstraction layer with interfaces for the different telecommunication features will be introduced. In addition a widget engine that makes these telecommunication interfaces available to its widgets will be presented. This will allow the rapid development of IMS applications for external developers and the combination of other Web 2.0 services with IMS features.

Keywords: Web 2.0, Widgets, Widget Engine, IMS, API, JavaScript, Google Android, Telecommunication, VoIP.

1 Introduction

Over the last few years there has been an identifiable tendency towards the convergence of classic telecommunication methods and Internet services. The availability of fast Internet connections with less delay is making new telecommunication applications that use the Internet possible. Services like Voice-over IP or instant messaging have gained wide acceptance from users. IMS is a strong candidate for a telecommunication network base of the future. It is specified by 3GPP and offers services like Voice-over IP calls, presence, location information and instant messaging. To access the IMS a client is usually required to be based among others in the Session Initiation Protocol (SIP).

In the past telecommunication companies had a conservative strategy when compared to web companies. Their services were traditionally not open and

C. Giannelli (Ed.): Mobilware, LNICST 0007, pp. 1–14, 2009.

could only be accessed by commercial clients with the provider's permission. The Internet community introduced another approach, as Web companies have tended to offer their own APIs to let the user build their own applications or services using the companies' data and services. This type of business benefits both sides: the user and the operating company. Users get new functionality to generate individual content while the vendor benefits from more customers. Most of these APIs are based on JavaScript, respectively Asynchronous JavaScript and XML (AJAX), because it is available in almost all modern browsers and it is not too complex. To reach various new developers those APIs have a simple structure. This enables rapid development of new ideas and the combination of different services from different companies: these are known as mash ups.

This paper will analyze the approaches to accessing telecommunication functions inside small Web 2.0 enabled applications called widgets. In order to run a widget, a runtime environment, called a widget engine, is required to host and manage the widgets. Widget engines are standalone applications and are written in a complex programming language. There is also another kind of widget called a browser widget, which is displayed directly inside a web page and is not stored and managed locally. Browser widgets will not be explored in this article.

Widgets usually use common web technologies like HTML, XML, CSS, JavaScript and AJAX. They can be created by external Web developers as these technologies are spread in the World Wide Web. This enables a big developer community. Due to the limitation to these Web technologies widgets have restricted functionality. A widget is similar to a Web page. It uses almost the same technologies and has the same limitations. HTML and XML are just markup languages without control options. Hence the features of the widgets are limited by the functions offered by JavaScript. For instance widgets have no direct access to the file system. Widgets can also not directly access the IMS because it is not possible to use SIP with JavaScript. Furthermore, due to security issues JavaScript does not have the permission to access functions or resources outside of its sandbox: other Web pages are the exception. All features which are not realizable using native JavaScript must be carried out by the Widget engine as the widget engine is not restricted in functionality. Thus the widget engine must provide extensions to use a widget's additional features. Other widget engines provide extensions for monitoring resource consumption or accessing device-specific capabilities such as calendar or contact list for instance.

This paper will introduce a telecommunication extension dealing with the opportunity to create "IMS Widgets", widgets that can use telecommunication features based on IMS. In order to ease the development process for web programmers not familiar with IMS the widget engine must provide an abstraction layer between Web technologies and the IMS world.

Section 2 will provide a brief introduction to widget engines currently available from other companies, and the W3C widget standard draft.

Section 3 will introduce a JavaScript API that provides telecommunication interfaces for services offered by IMS.

Section 4 describes a widget engine which offers telecommunication functions to its widgets by implementing the JavaScript telecommunication API.

2 Related Work

This section describes widget engines from big vendors which are widely used. A short introduction into the W3C widget recommendation and BONDI, an initiative for standardizing JavaScript extensions on mobile devices, will follow. Concluding with the Parlay X interfaces, a telecommunication API based on Web services.

2.1 Widget Engines

In the last few years a new kind of application has increasingly gained popularity: the so called desktop widgets. These widgets are small frames that provide services like weather forecasts or news headlines on the desktop. Widgets are interesting because they are easy to develop, deploy and use. Since widgets cannot run as stand-alone applications they require a widget engine in order to run.

Various companies have released their own widget engines. Konfabulator is supposed to be the first commercially successful widget engine. Apple's Dashboard which was integrated into the Mac OS X 10.4 operating system in 2005 was inspired by Konfabulator. In the same year Yahoo bought the Konfabulator and renamed it Yahoo Widget Engine. In 2006 Google introduced its Google Desktop Gadgets, another widget engine. Microsoft followed and integrated the Vista Sidebar, as a replacement for Active Desktop from Windows 98, which was also designed to display Web content on the client desktop but was not very successful. Another popular widget engine comes from Opera. Microsoft and Google use the term gadget which is in fact just another name for widget.

Almost all widget engines provide hardware device features like monitoring battery status or CPU usage but none of these widget engines have telecommunication features. Dashboard offers a Skype widget which is in fact able to use telecommunication features like VoIP or instant messaging. But this widget requires the normal Skype standalone-application to be installed on the system because the widget engine communicates with the standalone Skype application. Hence the communication part is not really integrated into the widget engine. None of these widget engines gives the developer the opportunity to access IMS services. Section 4 will introduce a widget engine that makes use of IMS services.

2.2 W3C Widget Standard

All widget engines named above are proprietary, meaning their widgets are not compatible. In order to fulfill the specific requirements, a developer who wants to deploy his widget on the most common engines has to write it several times. Nevertheless they have a great deal in common. All engines use common file formats like HTML, XML and JavaScript to ease development and to make integration of other Web 2.0 services comfortable. W3C has done a survey on the widget engines

currently available and they are now in the process of developing a widget standard [1]. The widget structure, as defined in the first draft, looks very similar to the Opera widgets' structure [2]. The actual draft does not yet include any information about JavaScript extensions offered to the developer. Such extensions are required to get access to additional features like hardware monitoring or telecommunication services from a widget. The widget engine, described in section 4, implements the actual W3C widget draft for packaging and configuration.

2.3 BONDI

While the W3C recommendation is focused on widget packaging and configuration, BONDI targets the functional opportunities available for web applications. BONDI is an initiative that is intended to address the problem of the non-interoperability of today's applications for mobile devices. The initiative was founded in mid 2008 by the Open Mobile Terminal Platform (OMTP), a forum of international market leading mobile operators and device manufactures.

Fragmentation of mobile platforms slows down the development of applications and in turn mobile Web usage. BONDI seeks to remedy that problem by proposing interfaces for typical telecommunication and device-specific services. The initiative sees the browser, and the underlying web rendering engine respectively, as the environment for future mobile applications. BONDI interfaces are proposed in a JavaScript API which can be used in normal websites as well as in widgets. These interfaces provide access to the key local capabilities of mobile handsets to ensure that web applications can access native functionality.

Release 1.0 of the BONDI JavaScript API is separated into 12 functional groups such as Messaging, Media Gallery, Personal Information (Calendar, Contacts), Location and Camera [3]. BONDI started integrating these features as JavaScript extensions into the browser engine of Windows Mobile.

BONDI's approach is similar to the JavaScript API introduced in this paper but BONDI interfaces are focused on device-specific capabilities rather than IMS telecommunication services.

2.4 Parlay X - Telecommunication API

Programming telecommunication networks is very complex. For this reason the Parlay Group was founded, an industry consortium of several telecommunication vendors. Its goal is to develop open standards for the telecommunication market. In 2003 Parlay released the Parlay X API, a collection of Web service interfaces to access telecommunication networks. These interfaces can be implemented by different vendors on different networks. Parlay X provides high level functions but is usually not intended for external Web developers from the Web community. Even though Web services are easy to use it is not that easy for a Web developer to integrate them into browser enabled Web 2.0 applications. A solution is required for combining IMS services with browser based Web technologies like AJAX [4]. The JavaScript Telecommunication API, introduced in the next chapter, provides an abstraction layer for the combination of Web technologies and IMS.

3 JavaScript Telecommunication API

This section introduces a telecommunication API, used to access the IMS inside of widgets. A brief introduction for the most relevant IMS services will be given.

3.1 API

All telecommunication features described in this paper are based on the IMS network. IMS is a telecommunication core network specified by 3GPP that is completely based on IP.

A new API for telecommunication features, offered by IMS, is proposed. The Telecommunication API is defined by JavaScript functions that can be used inside of a widget. JavaScript was chosen due its market diffusion and simplicity.

The JavaScript IMS functions are high level abstractions, meaning only one line of code is necessary for accessing an IMS service which might invoke several internal actions. Each function can either be called synchronously or asynchronously. Synchronously means that the script waits until the execution of the command is finished. Asynchronous calls execute the same command with a callback function as an additional parameter. The script goes further and the callback function is executed when the command is finished. Synchronous calls are easier to develop but have the disadvantage that the application might freeze if the invoked command takes a long time to return.

The API comes as a JavaScript file which must be included in all widgets that want to use telecommunication features. Section 4.4 describes how to include the API in a widget.

3.2 Telecommunication Services

IMS provides several kinds of telecommunication features. The aim of this API is to provide high-level interfaces for the most relevant services. All provided telecommunication functions are arranged into 8 service groups. Table 1 shows the proposed groups in alphabetical order.

First of all it is possible to initiate voice calls routed by IMS. There are three different categories of voice calls: peer to peer calls, third party calls and conference calls.

Table 1. JavaScript telecommunication API Service Groups

AddressBook	contains a set of functions for accessing user profiles via XDMS
Call	contains functions for outgoing and incoming voice over IP calls
ConferenceCall	a set of functions for voice connection of multiple users
Location	provides functions for getting location information
Messaging	contains functions for sending and receiving instant messages
Presence	provides functions for getting and setting presence state
SMS	allows the user to send a short message to a mobile phone
ThirdPartyCall	contains functions to start or end a third party call

Peer to peer calls connect two participants via a SIP protocol where voice information is streamed over an IP based network (Voice over IP). A gateway to the PSTN enables it to call using classic circuit-switched access as well. The API makes it possible to initiate and receive VoIP calls and to get information about a running VoIP call.

The second category of voice calls are **third party calls**. Third party calls also establish a voice connection between two participants from the point of view of a server which acts as the third party. A third party call expects two SIP addresses or phone numbers. After initiating the call participant 1 receives an invitation for an incoming call. When participant 1 has accepted the call participant 2 receives the notice of an incoming call as well. The voice connection is only established if participant 2 has also accepted the call. The control of the third party call lies on the server side, meaning the third party may also disconnect the call.

Another kind of call is the **conference call**. Conference calls are voice connections between at least two participants. The initiator of the conference may invite new participants and disconnect participants from a running conference.

A further telecommunication service offered by IMS is **instant messaging**. Instant messages are widely known from the Internet. The API makes it possible to send messages to the SIP address and to receive instant messages. This kind of service facilitates chat applications.

The IMS network includes a gateway to send **short messages (SMS)** to mobile phones. Thus it is possible with the telecommunication API to send a SMS.

The next functional group of the API is called **AddressBook**. It includes all functions that are necessary for managing personal contacts. Contacts are normally stored on the XML document management server (XDMS), which acts as an XML database for user management and configuration. User profiles may contain information like name, address, age, gender, telephone numbers or optional free text. Contacts can be arranged into different groups like “friends” and “teammates” for instance. The SIP address of each contact acts as its primary key.

Presence is another functional service group. The IMS network includes a presence server which contains the actual presence state for all registered users. The presence state is presented as one of the following strings: “ONLINE”, “OFFLINE”, “BUSY”, “AWAY”. The telecommunication API allows a user to set its own state, to get the actual state of the other users and to react to other users’ presence state changes.

The IMS also provides features for finding the **location** in terms of GPS coordinates. If the device that runs the client provides location information it can be accessed by the telecommunication API.

4 Widget Engine

This section introduces the new widget engine, which makes use of the telecommunication API, including its implementation on desktop computers and Google Android based mobile phones.

4.1 Requirements for a Widget Engine with Telecommunication Features

The widget engine cares about hosting, including managing and rendering the widgets. It should run on different platforms: Windows XP, Windows Vista, Linux, Windows Mobile and Google Android. Widgets should be portable, meaning a widget that was written for the desktop engine should run on other platforms too.

The widget engine is compliant to the current W3C draft standard [2]. That is to say a widget uses only common file formats known from the web. For this purpose the widget engine must include an interpreter for HTML and JavaScript code. The goal of this paper is to introduce telecommunication features into widgets. Hence the widget engine must additionally provide interfaces for telecommunication features to the widgets by offering the JavaScript functions of the Telecommunication API described in section 3.

Any incoming notification message must be delivered to the appropriate widget. This is the situation in which the widget engine receives an incoming call, an incoming instant message or a presence change. The web developer may define JavaScript functions that will be invoked when such an incoming notification occurs. If no widget is open while an incoming call arrives the user may define a standard widget which will be opened automatically when a call arrives.

4.2 Architectural Design

The widget engine comes as a stand-alone application which is installed on the terminal. On startup it logs in to the configured IMS network automatically. Each widget is packaged as a zip file containing no more than the common Web file formats. Widgets can be installed from external media storages or Web sites into the engine. They are stored in a local directory as part of the widget engine. The graphical interface of the engine differs depending on the operating system.

Figure 1 illustrates the interaction of the components. IMS access is realized via MONSTER, a standards-compliant client framework developed at Fraunhofer FOKUS that implements JSR 281 [6][7]. The acronym MONSTER stands for Multimedia Open Internet Services and Telecommunication Environment, an extendible framework that provides communication interfaces for accessing the IMS network. Due to its Java implementation, MONSTER is applicable on multiple platforms. MONSTER was implemented and tested in the Open IMS Playground, a Fraunhofer FOKUS IMS environment, but is applicable on other IMS networks too [10].

When a widget invokes a JavaScript function of the telecommunication API, the widget is responsible for doing the mapping to the appropriate MONSTER interface. In the other direction MONSTER provides listener services which are fired when an incoming notification arrives. The incoming notification will be delivered to the appropriate widget by the widget engine. The challenge is how to map the functions of the JavaScript API to the underlying IMS services. Some

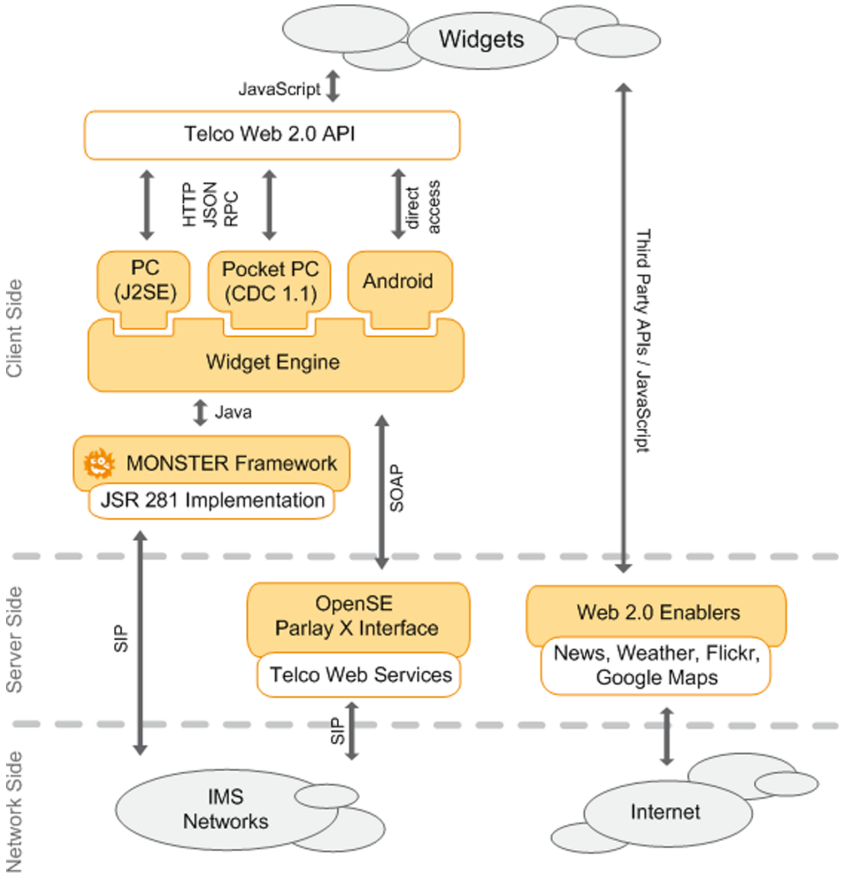

Fig. 1. Widget engine with interacting components

features, like short messaging for instance, are executed by calling Web services offered by OpenSE, a server based implementation of the Parlay X interfaces described in section 2.3. Both MONSTER and OpenSE access the IMS via SIP protocol. Because JavaScript has no built-in access to Java and certainly no possibility of using SIP, the widget engine provides a way for communicating between Java and JavaScript.

4.3 Implementation of the Desktop Widget Engine

Each widget runs in a dedicated window, which in fact contains a lightweight Web browser to render the widget. The desktop version allows drag and drop of the widgets on the screen. Moreover it allows multiple widgets to be opened in parallel.

The implementation uses Java Standard Edition, thus it is possible to use the same code on Windows and Linux. The Standard Widget Toolkit (SWT)[1], developed by the Eclipse foundation, was chosen as the GUI toolkit [11]. SWT is an open source GUI toolkit, designed to provide user interface elements based on the operating system. It meets the requirements in terms of portability, performance and the GUI facilities provided. Because it uses native user interface facilities from the operating system, SWT is much more efficient than Swing or AWT. SWT offers a Web browser element, which is able to interpret HTML, JavaScript and CSS files and to display image files like any other modern browser.

The SWT browser element uses Xulrunner, a runtime package that can be used for the interpretation of HTML and JavaScript code [7]. Xulrunner was developed by Mozilla and is available for Windows and Linux. It uses the Gecko engine, known from the Firefox Web browser. On Windows systems it is also possible to exchange Xulrunner with Internet Explorer.

Two different approaches were considered for the mapping from JavaScript to Java. Both use Jabsorb, an open source Java tool for Java-JavaScript interaction [8]. Jabsorb provides methods for marshalizing Java objects into JavaScript Object Notation (JSON) strings. JSON is a text-based format used to exchange data which is similar to XML but with a simpler structure and less overhead. The invocation of Java methods is done by JSON-RPC, a text based protocol for calling on remote procedures. These JSON strings can be sent from JavaScript to Java and vice versa. The receiving side unmarshalizes the JSON string into a Java or JavaScript object. The two approaches for the JavaScript to Java mapping differ in the way in which they transmit the JSON strings.

The first approach uses the browser status bar as a buffer. When JavaScript wants to invoke a Java method a JSON string is written on the browser status bar. The user will not notice that because the status bar is invisible. In Java the browser element has an onChange listener method for the status bar which is invoked when the status text changes. The JSON string on the status bar is unmarshalized by Jabsorb into Java objects / method calls and the appropriate Java method will be invoked. This approach is efficient because the delay between method invocation and execution is very small. The disadvantage of this kind of communication is that it does not allow synchronous calls. JavaScript has no built-in support for multithreading. Thus it is not possible for the JavaScript code to wait until Java returns.

The second approach is based on a local lightweight Web server with a small memory footprint. The JSON string is transmitted via AJAX to the local HTTP server. The server receives the string and sends it to Jabsorb which unmarshalizes it, executes the appropriate Java method and sends a reply to JavaScript. This kind of communication requires a local server. A lightweight HTTP server has been implemented and integrated into the widget engine for this purpose. Due to the server request this kind of communication suffers a longer delay than the first

[1] Please note that the term widget is used in another context when talking about GUI toolkits. A GUI toolkit widget is a graphical element for user interaction like an input field or a check box.

approach. The advantage of this approach is the opportunity to use synchronous calls. AJAX makes use of the XML.HTTP request object which is able to invoke both asynchronous and synchronous calls.

Incoming notifications can be delivered directly to JavaScript by using the browser.execute() command which is included into the SWT browser. In this way incoming messages may cause the browser element to invoke the appropriate JavaScript function inside a widget. A widget must register for incoming messages since not all widgets are interested in knowing when an incoming message arrives. Therefore the widget developer can write a JavaScript function called init(), which is invoked on startup after the widget finishes loading.

Each widget using the telecommunication API on the desktop version must include two JavaScript files in its initial HTML file. The first .js file manages the JavaScript to Java mapping and the second one contains the telecommunication interfaces.

4.4 Mobile Version Using Google Android

Android is a new open-source platform for mobile devices introduced by Google at the end of 2007. It was released in the form of a development toolkit including an emulator for testing new applications. Android is based on Linux Kernel 2.6 and the Dalvik virtual machine, a JVM with some additional features like the built-in GUI toolkit. Indeed it has some restrictions as well. Due to security issues an Android application cannot access files outside of its own application directory for instance.

Android applications are usually composed of multiple so called activities. An activity is mostly a single screen that interacts with the user. Every activity has its own complex lifecycle including a large set of controlling points. When writing their own widget a developer does not need to be concerned about Android's programming model. They can use normal JavaScript in the same way as they do when creating a Web site.

The Android version of the widget engine uses the same MONSTER communication interfaces to access the IMS as the desktop version. For the GUI no extra toolkit like SWT is required because Android has built in GUI elements that can be used. The widgets are interpreted by an element called WebView, which is part of WebKit, a library included in Android. WebKit is known as the rendering engine of Apple's Safari browser. This WebView element replaces the SWT browser of the desktop version. Because of the smaller display the Android version of the widget engine will not show more than one widget at the same time. The engine itself is realized using the GUI element "gallery" of Android. The end-user may load new widgets from an SD-card into the engine or Web and remove old widgets. Every icon in the gallery represents a stored widget.

The communication between Java and JavaScript is very efficient on Android. WebView elements provide the opportunity to expose Java objects to JavaScript, whose methods can be accessed directly inside the JavaScript code. In the opposite direction Java can invoke JavaScript functions inside a WebView by calling WebView.loadUrl("javascript: code"). Due to the different mapping from Java to

JavaScript, widgets loaded into Android have to include another JavaScript file for the telecommunication API than the desktop version. The following section describes this process. Nevertheless the telecommunication API for Android provides the same interfaces but uses a different system for mapping to the widget engine.

4.5 IMS Widgets

As defined in the W3C draft, every widget contains at least a config.xml for the meta data and a starting HTML file, usually called index.html [2]. As an extension of the widget engine the config.xml must include a link to an image file which describes the shape of the widget.

5 Evaluation

The aim of this work is to explore widgets' IMS telecommunication features. In order to prove their usability, portability and ease of development an example widget was created for every service group of the telecommunication API. This section shows some of these example widgets using the new widget engine, including telecommunication services.

5.1 Call Widget Example

Figure 2 shows an example call widget. When inserting a number or SIP address into the input field and clicking the green button on the left a new connection will be established. Another use case appears when an incoming call arrives and the user clicks on the green button. In this case the incoming call will be answered, instead of initiating a new call. The user has the power to cancel the call or see viewing information about the call by pushing the buttons on the bottom of the widget.

Each widget may contain an init function, which can be written by the developer. This function is executed after the widget has finished loading. The

Fig. 2. Call Widget

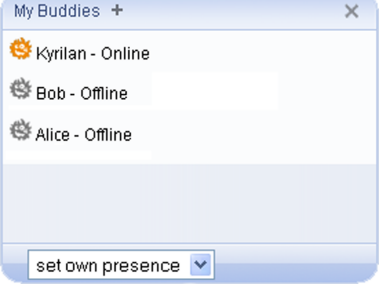

Fig. 3. Address Book / Presence Widget

init function is meant to define what happens when incoming events, like an incoming call for instance, occur.

The following JavaScript code shows the functional part for the green button in order to demonstrate the simplicity of the API. If the status equals "Incoming" the ring tone stops and the incoming call is answered. answerCall function is invoked asynchronously with a callback function as input parameter while makeCall invokes the Java method synchronous. This shows the different options for invoking the telecommunication API.

```
function connect() {
 if (actualStatus=="Incoming") {
  /* when call is arriving answer call */
  var callBackFunction = function(result) {
   stopRing(); /* stops the ringtone sound */
  }
  FOKUS.Call.answerCall(callBackFunction);
 }
 else {
  /* when number entered initiate call */
  var tel = document.getElementById("tel").value;
  if (tel!="") FOKUS.Call.makeCall(tel);
 }
}
```

The high-level function makeCall executes a lot of consecutive actions. First of all the request is sent to the widget engine which executes the appropriate Java method in MONSTER. MONSTER sends a SIP invite packet to the P-CSCF of the client's IMS network. The P-CSCF redirects the invitation to the I-CSCF which looks up the called user's S-CSCF in its HSS. Subsequently the S-CSCF forwards the invitation to the called user's client. MONSTER also manages the authentication, authorization and accounting (AAA).

The invocation of Java methods by JavaScript causes a delay of less than 10ms when using the implementation based on status bar communication. The local server based communication requires no more than 20ms on a regular desktop computer.

As can be seen the invocation of the IMS service is very comfortable using the JavaScript telecommunication API. A widget developer does not have to worry about the complexity behind the JavaScript functions. The high-level functions hide all consecutive actions. The use of JavaScript allows the integration of other Web 2.0 services.

5.2 Presence and Address Book Example Widget

The second example widget based on the widget engine combines the address book service with the presence service. It displays all buddies entered in the user's address book and shows their actual presence state. Inside of its init function the widget invokes the following actions:

1. read all entries from the user's address book
2. get presence information for every entry
3. subscribe to be notified of presence changes for all entries
4. set callback function for incoming presence state change
5. display all entries with their presence state

When any of the buddies changes his presence state the widget gets noticed immediately and displays the new presence state. With the select box on the bottom of the widget a user may publish its own presence state to the IMS' presence server. The whole widget implementation requires about 40 lines of JavaScript code and less than 30 lines of HTML code. Design is defined in an extra stylesheet file.

5.3 Widgets on Android

The same widgets deployed on the desktop widget engine will also run on Android. This enables rapid development of Android IMS applications without knowing anything about Java or the Android code model. Only some lines of HTML and JavaScript must be written to create an IMS application on Android.

Figure 4 shows the widget engine on an Android emulator. The GUI of the widgets was adapted in order to take the issue of fingertip handling into account. The functional part is the same as for the desktop version. On the top you can see the engine, where every icon represents a widget. The actual opened widget is displayed below. The figure on the left shows the presence / address book

Fig. 4. Widget Engine on Android Emulator

widget which was presented in section 5.2 for the desktop version. The figure in the middle shows a chat widget which allows to send and receive instant messages based on the IMS network. The figure on the right shows an example widget for sending an SMS.

6 Conclusion

The widget engine described in this paper makes it possible to develop IMS client applications in a fast and easy way. The telecommunication API provides a layer of abstraction between the IMS and the widgets. Using this high-level abstraction layer, a developer with only web programming experience can easily handle the functional complexity behind IMS. This allows the creation of customized IMS clients for a wide range of users. Due to the fact that widgets make use of Web technologies, exposing IMS based telecommunication services to Web 2.0 enabled applications becomes an exploitable opportunity. It has been proven that the developed widgets are easy to create, are highly portable and run on various operating systems and hardware platforms.

Future work will enhance the telecommunication API with services for music and video streaming that can be used inside of Web applications. These features are already supported by the MONSTER framework and it is planned to make them available to widgets. At the moment Android restricts VoIP calls because Google has disabled audio streaming. Another open issue is widget to widget communication. Widgets can interact with the widget engine but cannot interact with each other yet. Future work will concentrate on this issue. As another next step the widget engine will be ported to Windows Mobile.

References

1. Widgets 1.0: The Widget Landscape, W3C (April 2008)
2. Widgets 1.0: W3C Working Draft, W3C (April 2008)
3. BONDI – Interfaces Requirements Version 1.0, OMTP (February 2009)
4. Hughes Systique Corporation. Telco-ajax: A concept approach at bringing legacy telecom application servers to web 2.0. Whitepaper (Feburary 2008)
5. Parlay X Web Services, Parlay Group, Version 3.0 (June 2007)
6. JSR 281 IMS Services API, Java Community Process, Ericsson (July 2008)
7. Design of a Coherent Mobile Multimedia Framework for Convergent Services, Alberto Diez Albaladejo, Alin Murarasu, Thomas Magedanz, TU-Berlin (2008)
8. Jabsorb 1.3, Open Source (May 2008)
9. Xulrunner 1.8.1, Mozilla (March 2008)
10. IMS Playground, Fraunhofer FOKUS, `http://www.open-ims.org`
11. Standard Widget Toolkit, Eclipse Foundation, `http://www.eclipse.org/swt/`