D37.161

3.16a a MOBiNET Dashboard development

(release 1)

Work package: 3.7
Version number: Version 0.6
Dissemination level: PU
Date: 09/07//2014

7th RTD Framework Programme
Directorate General for Communications Networks, Content & Technology
Cooperative Systems for energy efficient and sustainable mobility (FP7-ICT-2011-6.7)
Contract Type: Collaborative project
Grant agreement no.: 318485

Version Control

Version history			
Version	**Date**	**Main author**	**Summary of changes**
0.1	04/06/2014	Benjamin Hebgen (NEC)	Proposed TOC
0.2	10/06/2014	Lars Mikkelsen (AAU)	Widget Overview added
0.3	13/06/2014	Andreas Rickling (DLR)	Widget Development added
0.4	18/06/2014	Benjamin Hebgen (NEC)	Architecture Update + Merge
0.5	20/06/2014	Benjamin Hebgen (NEC)	Rave Overview
0.6	25/06/2014	Benjamin Hebgen (NEC)	Intro + Concluding chapter
	Name		**Date**
Prepared	Benjamin Hebgen (NEC)		26/06/2014
Reviewed	William Lauer (Pluservice), Gerardo Daalderop (NXP)		03/07/2014
Authorised	Paul Kompfner (ERTICO)		10/07/2014

Circulation	
Recipient	**Date of submission**
European Commission	11/07/2014
Project partners	11/07/2014

Authors

Benjamin Hebgen (NEC)

Andreas Rickling (DLR)

Lars Mikkelsen (AAU)

Table of contents

Abbreviations and definitions

Abbreviation	Definition
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ATOM	Atom Syndication Format. An XML based format
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
OAuth	An open standard for authorization
OpenID	An open standard for authentication
OSDE	OpenSocial Development Environment
W3C	World Wide Web consortium
XML	Extensible Markup Language

1 Introduction and Overview

This document provides an updated version of the architecture of the MOBiNET Dashboard described in D31.2. Compared to the Initial Concept and Architecture Deliverable this deliverable will give a more detailed look into the actual implementation. In order to give an overview we show an updated high level architecture describing the basic functionality of the Dashboard. Furthermore we describe the actual implementation of the architecture and the used tools. Additionally we provide an overview on how to develop widgets for the Dashboard as well as an overview of the already developed widgets. Additionally we describe the integration with the MOBiNET Identity Manager. The document finalizes with an outlook to release 2.

1.1 Open ID and OAuth

Within this document the terms OpenID and OAuth are used hence we give a quick overview over these two standards.

1.1.1 OpenID

OpenID is an open and standardized protocol which provides a way to authenticate users across websites using a third party service. This means a user can use one account to log into several unrelated websites. In order to do so users have to create an account at an OpenID identity provider. If a website trusts this OpenID provider the user can use his OpenID account to log into the website.

1.1.2 OAuth

OAuth is an open and standardized protocol for authorization. OAuth provides a way that let clients access server resources on behalf of a resource owner. Using OAuth resource owners can delegate access to their resources without sharing their personal credentials. OAuth basically allows an authorization server to issue access tokens to third-party clients. The client can now use the access token to access protected resources on a resource server.

2 Architecture Update

For the release 1 of the MOBiNET Dashboard the architecture was redesigned in as shown in Figure 1. The Dashboard consists of two main components, a widget container and a data analytics server. The widget container provides various widgets to communicate with the MOBiNET Service Directory and the MOBiNET App Directory. Additional third party widgets can easily be developed if required.

A major difference to the architecture in D31.2 is that widgets communicate directly with their service backend instead of going through a message bus. This change provides flexibility and does not force the service provider to integrate with a message bus but allows him to provide already existing interfaces. This not only allows the easy integration of new widgets for new services but also avoids scalability issues on the message bus.

In order to communicate with the MOBiNET Identy Management the Dashboard supports an OpenID based login. The widgets within the container are able to use the login information to check for authorization. Authorization is handled using OAuth.

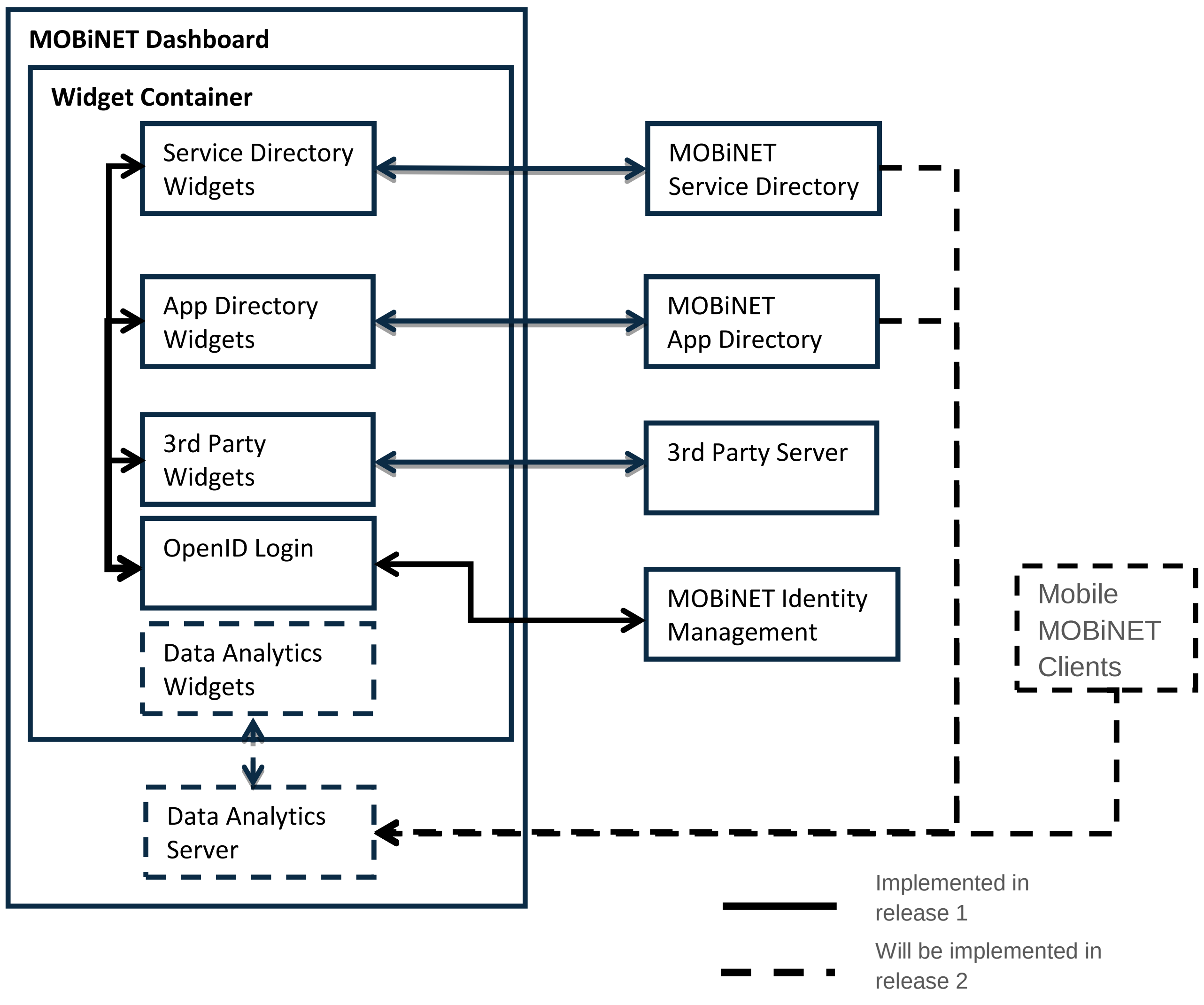


Figure 1 High Level Architecture Dashboard

In order to provide accurate analytics about the user behavior the Dashboard will contain a data analytics server. This server will interface with the Service Directory and the App Directory to gather information about the service and app usage. Additionally mobile MOBiNET Clients can send their local sensor information to the data analytics server. This additional sensor information can be used to derive contextual information about the user environment.

As shown in Figure 2 the widgets within the Dashboard have the capability to communicate with each other over a communication bus. The communication bus provides a publish and subscribe system and supports topic based subscriptions. This will allow an easy integration of new widgets. For the actual implementation we have chosen the Apache project Rave as a basis, which is described in the chapter 3.

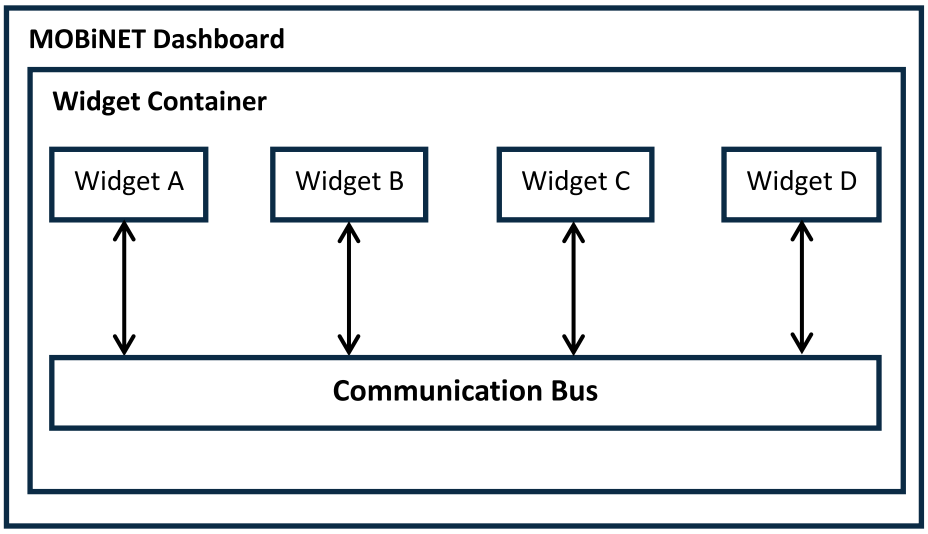


Figure 2 Inter Widget Communication

3 Apache Rave Overview

Apache Rave is used as widget container in the MOBiNET Dashboard. As shown in Figure 3 Rave uses two other Apache projects, Wookie and Shindig, to support W3C Widgets as well as OpenSocial Gadgets. Additionally Rave provides a portal which not only allows an easy composition of various widgets but as well provides OpenID and OAuth integration.

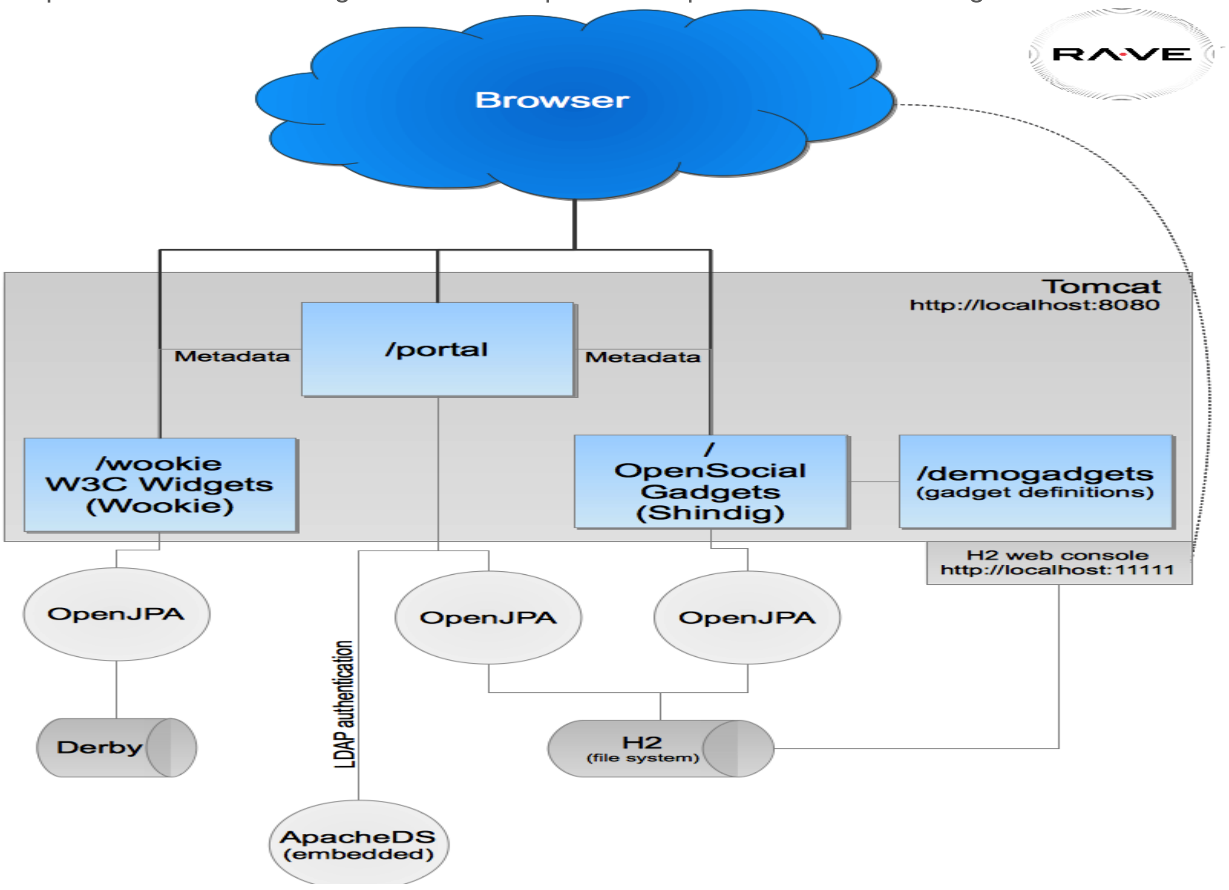
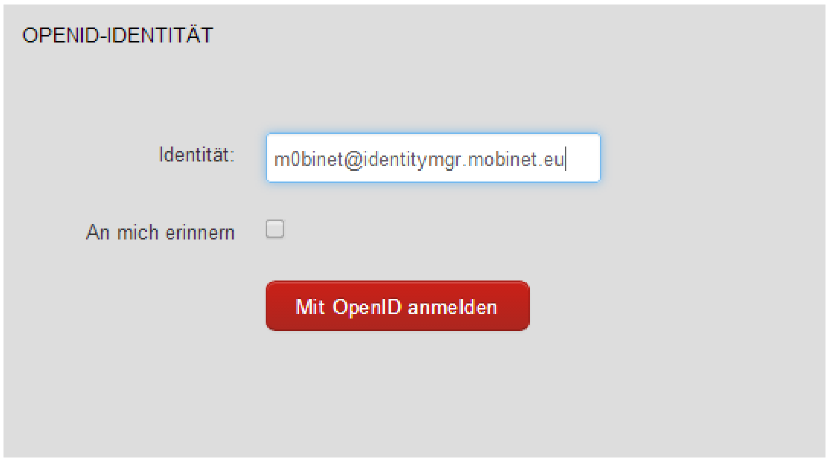


Figure 3 Apache Rave Overview source: http://rave.apache.org/

Using OpenID the Dashboard integrates with the MOBiNET Identity Manager for authentication. Authorization requests for different services will be handled via OAuth. OAuth integration is support by the OpenSocial Gadgets as well as W3C Widgets.


OPENID-IDENTITÄT

Identität: m0binet@identitymgr.mobinet.eu

An mich erinnern ☐

Figure 4 Dashboard OpenID Login

As shown in Figure 5 OpenSocial Gadgets have their own publish and subscribe bus while W3C Widgets have to use a third party communication bus. In order to compensate for this Wookie provides Wave as communication bus. Communication between OpenSocial Gadgets and W3C Widgets is also possible using a third party communication bus. This can also be achieved with the provided Wave bus.

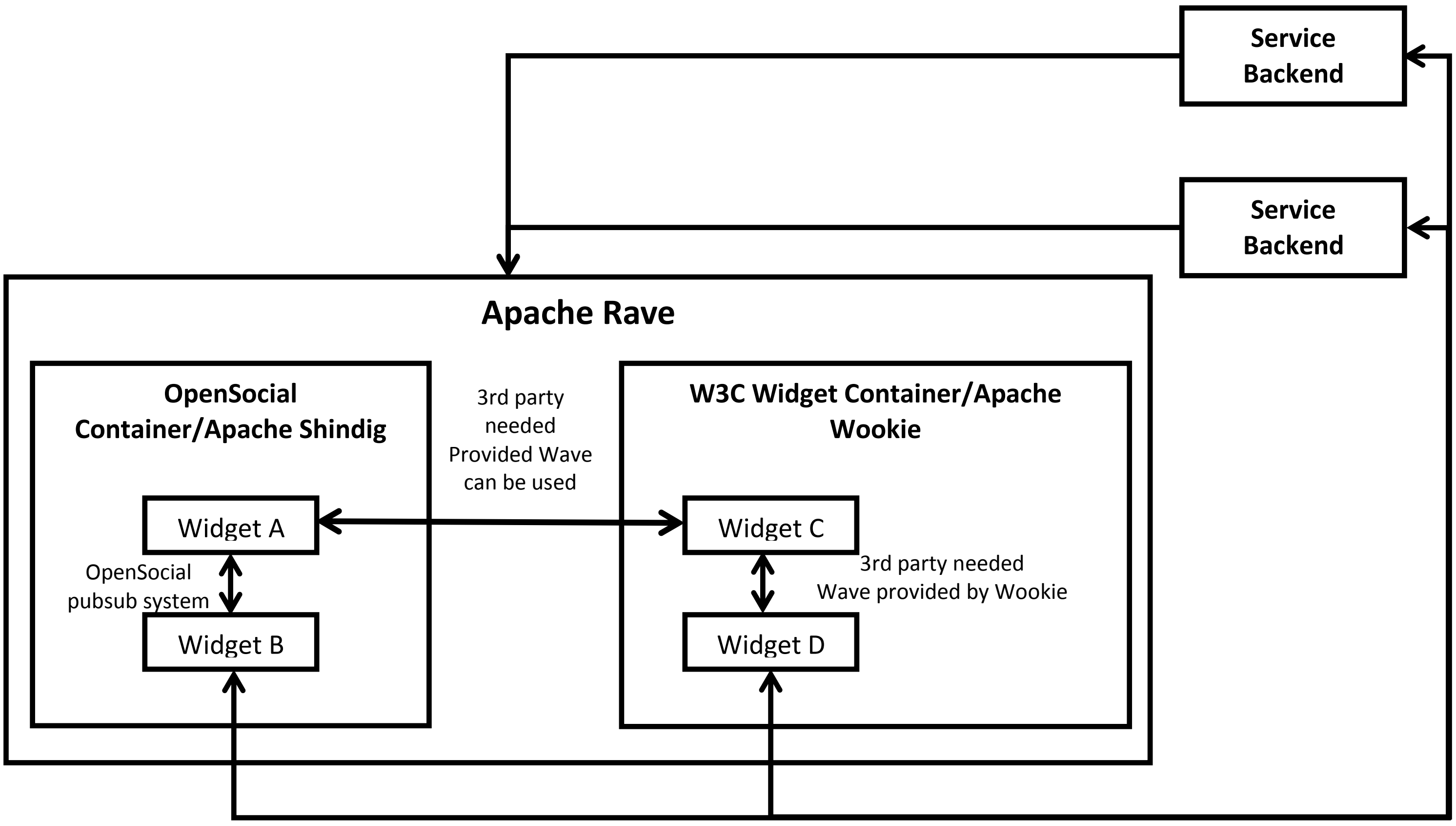
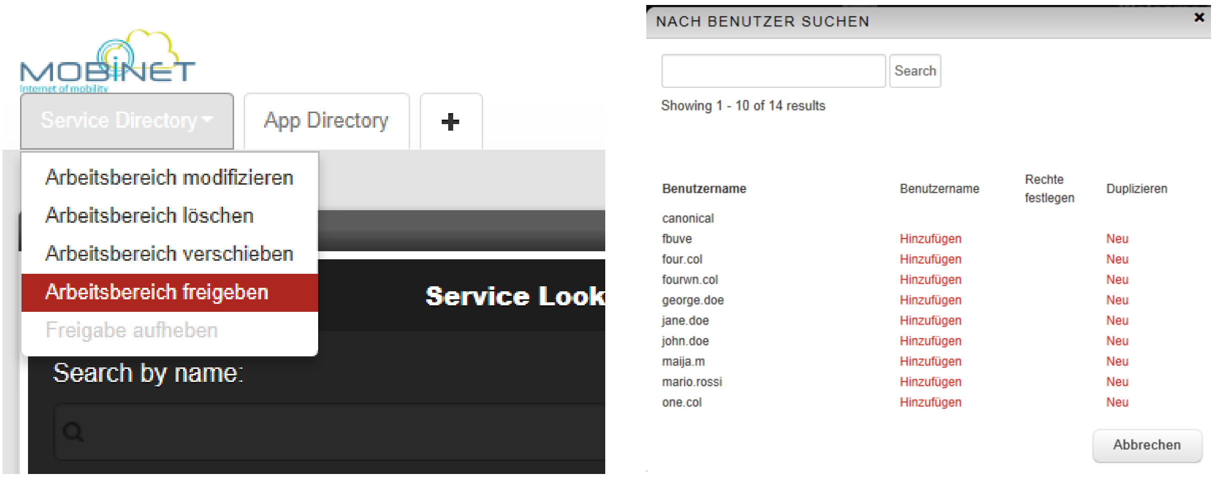

Service Backend
Service Backend
Apache Rave
OpenSocial Container/Apache Shindig
3rd party needed Provided Wave can be used
W3C Widget Container/Apache Wookie
Widget A
Widget B
OpenSocial pubsub system
Widget C
Widget D
3rd party needed Wave provided by Wookie

Figure 5 Interwidget Communication within Apache Rave

3.1 Identity Manager Login Widget

The communication between the Identity Manager and the Dashboard is established via OpenID. On the Dashboard side an integration of OpenID is provided by Apache Rave which already implements the OpenID standard. Using OpenID a new user to the Dashboard will be directed to the Dashboard account creation page. The new created account is associated with the OpenID account. In order to gain access to a specific section of the Dashboard an administrator has to unlock specific parts of the dashboard to specific users.



Showing 1 - 10 of 14 results

Benutzername	Benutzername	Rechte festlegen	Duplizieren
canonical			
fbuve	Hinzufügen		Neu
four.col	Hinzufügen		Neu
fourwn.col	Hinzufügen		Neu
george.doe	Hinzufügen		Neu
jane.doe	Hinzufügen		Neu
john.doe	Hinzufügen		Neu
maija.m	Hinzufügen		Neu
mario.rossi	Hinzufügen		Neu
one.col	Hinzufügen		Neu

Figure 6 Sharing workspaces in Rave

4.Dashboard Widget Development

The Apache Rave mashup engine supports two different kinds of widgets, the OpenSocial and W3C widgets. Both can be simply developed with standard web techniques based on HTML, CSS and JavaScript. They can work and communicate to each other in the rave engine. The following two subsections describes the functionality of their approaches.

4.1 OpenSocial

OpenSocial ist a set of APIs for building social applications that run on the web. OpenSocial provides a common API that can be used in many different contexts. OpenSocial widgets uses a declarative XML Syntax and can be interpreted by a supported gadget server to be embedded into various contexts. The embedded widget is called a widget container.
This is for example a standalone web page or a web application. Typically a widget container is able to display widget. These widgets follows the OpenSocial API and a gadget server has to provide the core features. Shinding is an open source project that provides a reference implementation of the OpenSocial container (link). Working out how to access and share data between applications can be tricky. OpenSocial provides a REST and RPC API through which OpenSocial-compliant applications and containers interact with each other, transmitting user data, friend lists and activities. The protocols support a variety of data exchange formats, including JSON, XML and ATOM. OAuth allows users to authorise access to data stored in social networks.

4.2 W3C

Like the OpenSocial Gadgets, the W3C widgets based on the standard web techniques HTML, CSS and JavaScript are mini-applications that can be run with Apache Rave. These kind of widgets are standarized by the World Wide Web Consortium (W3C). A widget is here specified as packaging format and metadata for a piece of software called as packaged apps or widgets.
Widgets relys on a Zip file as packaging format where an XML file declares metadata and the configuration parameters for a widget.
This specification is a part of the Widgets family of specifications

4.3 Motivation to use OpenSocial gadgets

The widgets for the dashboard component are currently developed with the OpenSocial standard 0.9. To build a widget there exist an integrated development environment to implement OpenSocial applications. The OpenSocial Development Environment (OSDE) developed by google is provided as an Eclipse Plug-in. In this environment you develop an OpenSocial application just with HTML, JavaScript and CSS technique.

The main motivation to use OpenSocial is a standard social vocabulary allowing any social app to talk to any other social app with minimal integration and cost. In the following are some other reasons listed to use this approach:

- Aggregation – ability to run multiple applications originating from different domains on the same Web page and allowing inter-widget communication. Cross Domain – avoiding cross domain issues by using OpenSocial container and proxy mechanism.
- Technology – HTML5, JS, CSS/LESS
- Extensibility -
- Anyone with a basic knowledge can create widgets and add them to the portfolio.
- Platform-wise, Cloud Portal development teams continuously extend and provide new OpenSocial features for the user's benefit.
- Customizability –any user can customize and design the widget by using basic CSS or advanced LESS capabilities.
- Backward Compatibility – the ability to easily wrap/convert existing application to a widget.
- Deployment – You can deploy and host the widget anywhere. Be it HANA Cloud Platform or Dropbox.
- The pubsub-2 feature provides a publish-subscribe mechanism for inter-widget communication.

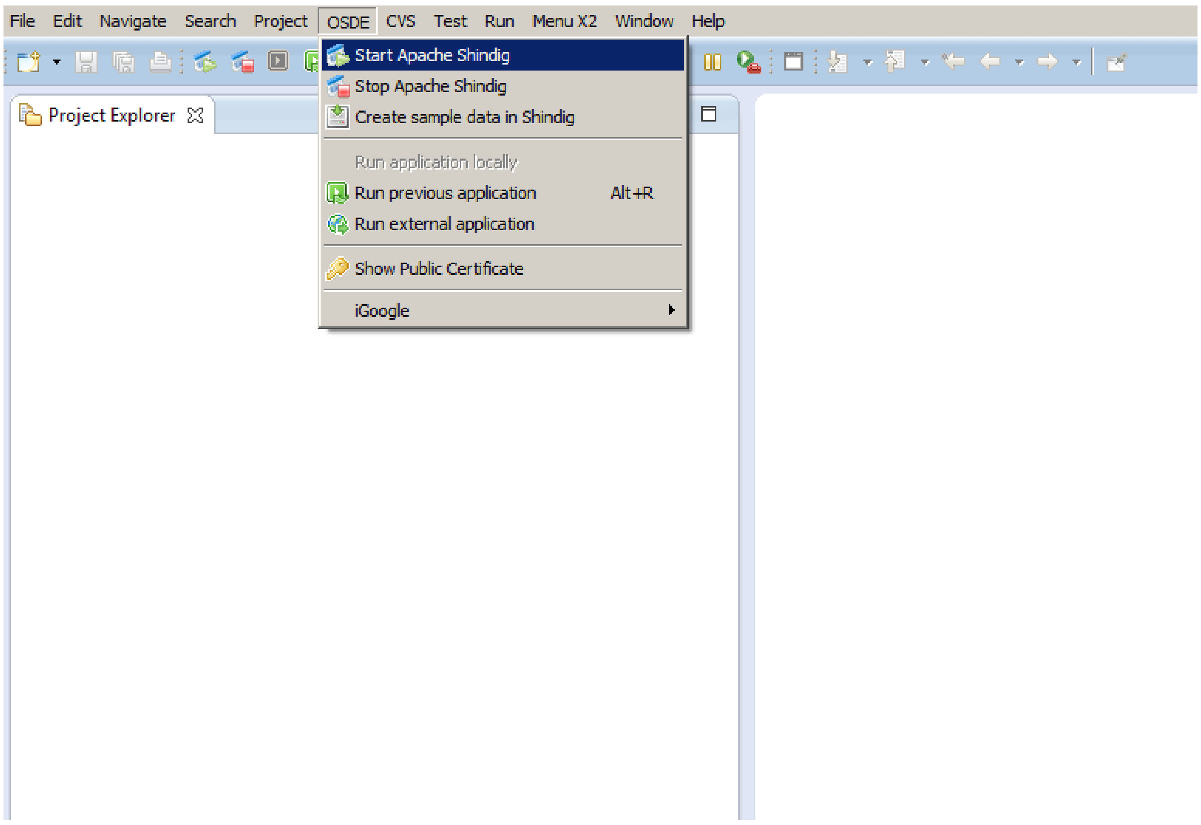
In the end it is easy to create a OpenSocial widget with a well-known development environment like Eclipse. The following section describes a sample development of a widget.

4.4 Sample Implementation of a OpenSocial gadget

With the OSDE Plugin ist is possible to build OpenSocial applications directly within Eclipse. OSDE enables the possibility to develop OpenSocial applications for both the client (gadget) and on a server using the Java RESTful client libraries. The following features can currently used with the plugin:

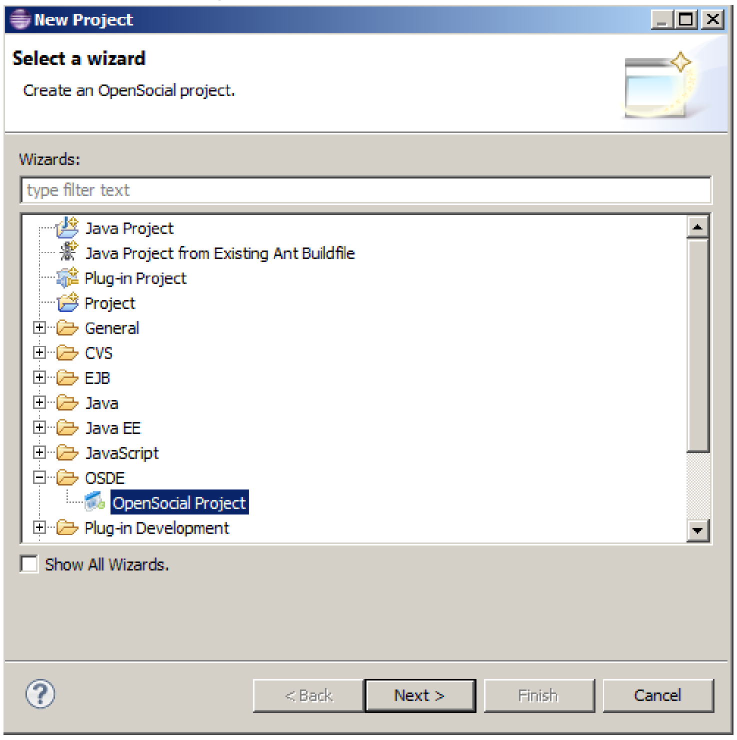
- A built in Shindig server for local testing and debugging.
- An integrated database based on the Java H2 for storing and managing social data.
- Simple project and gadget code generation wizards to quickly generate your OpenSocial application projects and application code.
- A multi-paned gadget spec editor that facilitates productive gadget development.
- A new "OpenSocial" Eclipse perspective that provides enhanced editing and debugging with easy access and control of the local Shindig and social database.
- An OpenSocial REST Java client wizard that can quickly generate a new project along with sample code that uses the OpenSocial RESTful Java client library to connect to the local OpenSocial server.

1. Launch the integrated shindig server.



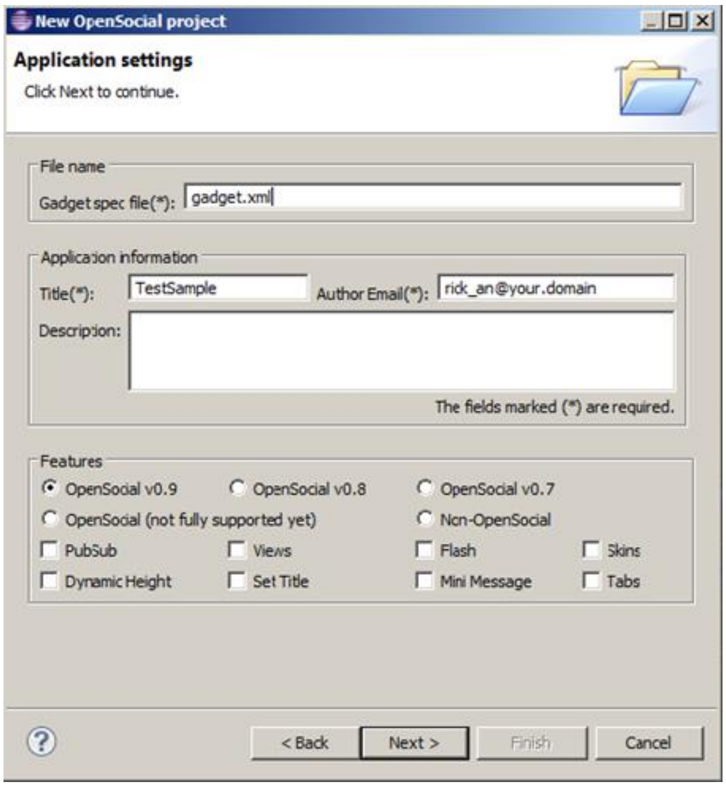
2. Now create the sample data in Shindig. Navigate to OSDE → Create sample data in shindig. Now we have some social data created.

3. Create a new OSDE project. Select in the menu File → New → Project and select the Point “OpenSocial Project” in the folder OSDE.



On the next Wizard Step enter the name of the Project and click on next.

4. Now there is a window called "Application settings"



New OpenSocial project

Application settings

Click Next to continue.

File name

Gadget spec file(*): gadget.xml

Application information

Title(*): TestSample

Author Email(*): rick_an@your.domain

Description:

The fields marked (*) are required.

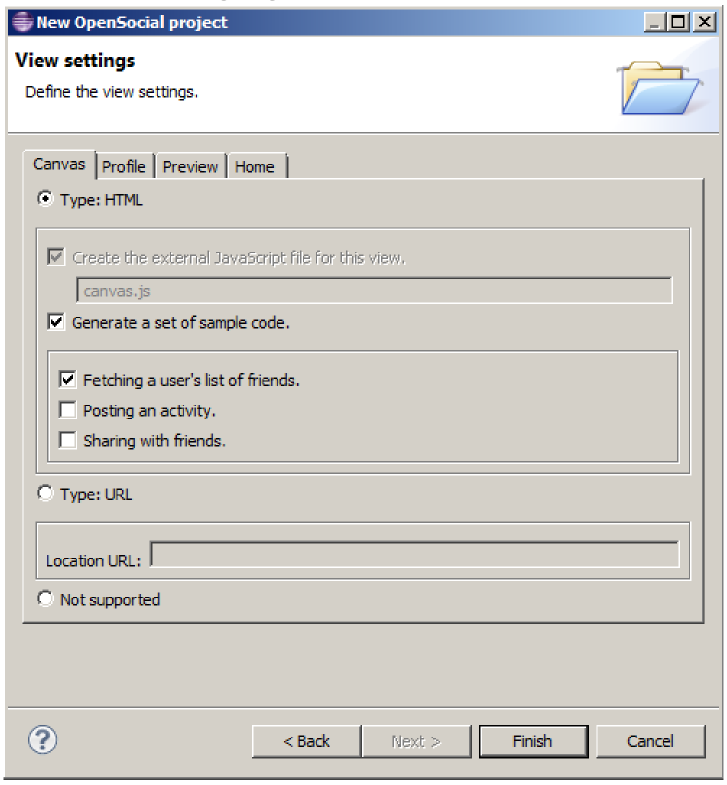
Features

- (•) OpenSocial v0.9
- ( ) OpenSocial v0.8
- ( ) OpenSocial v0.7
- ( ) OpenSocial (not fully supported yet)
- ( ) Non-OpenSocial
- [ ] PubSub
- [ ] Views
- [ ] Flash
- [ ] Skins
- [ ] Dynamic Height
- [ ] Set Title
- [ ] Mini Message
- [ ] Tabs

Here you can enter the app title and the author as well as the OpenSocial version.

5. The next step allows for the creation of 'views' of the application. These include 'canvas', 'profile', 'preview' and 'home'. For a simple app, we can create both a 'canvas' view and a 'profile' view, which is consistent with many OpenSocial containers.
For both a canvas and profile view, we can create them of type HTML, and select the option to generate JavaScript into a separate file, along with an initial 'init()' function to be

called when the gadget renders.



View settings

Define the view settings.

Canvas | Profile | Preview | Home

(●) Type: HTML

[✓] Create the external JavaScript file for this view.

canvas.js

[✓] Generate a set of sample code.

[✓] Fetching a user's list of friends.

[ ] Posting an activity.

[ ] Sharing with friends.

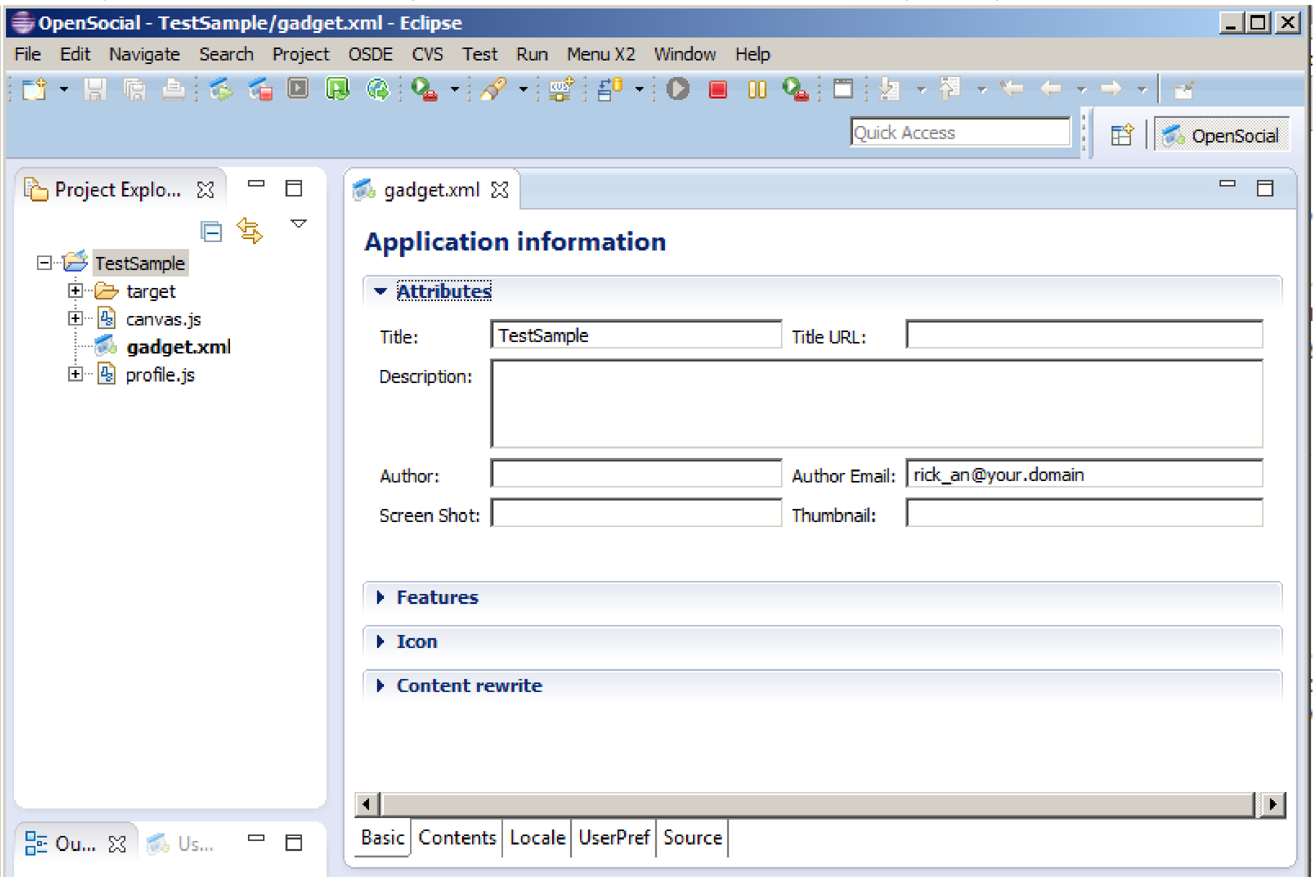
( ) Type: URL

Location URL:

( ) Not supported

Select the same settings on the Profile tab and click finish.

6. The Project will be created and you can see the folder structure in Project Explorer.



OpenSocial - TestSample/gadget.xml - Eclipse
File Edit Navigate Search Project OSDE CVS Test Run Menu X2 Window Help
Quick Access
OpenSocial
Project Explo...
gadget.xml
Ou...
Us...
Basic Contents Locale UserPref Source

TestSample
- target
- canvas.js
- gadget.xml
- profile.js

Application information

Attributes

Title:	TestSample	Title URL:	
Description:			
Author:		Author Email:	rick_an@your.domain
Screen Shot:		Thumbnail:	

Features

Icon

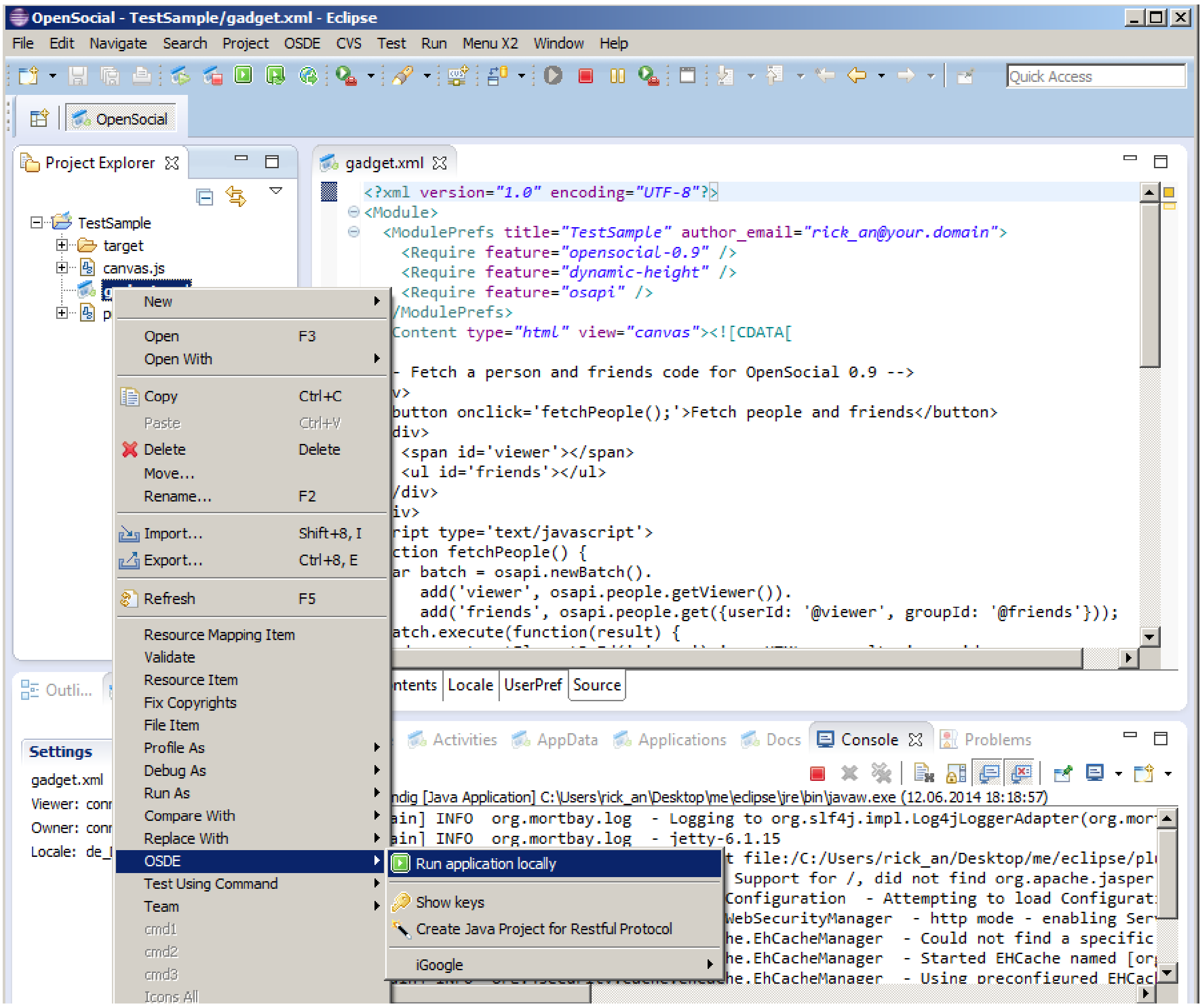
Content rewrite

In the “Basic” view of the editor you can change comfortable the gadget settings without changing the source code. In the “Source” view of the gadget.xml you see all the settings added with HTML code and JavaScript of this sample OpenSocial gadget.

7. Now run the Project with right click on gadget.xml → OSDE → Run application locally.

   On the next window “Run application locally” just click “OK” to run the gadget.

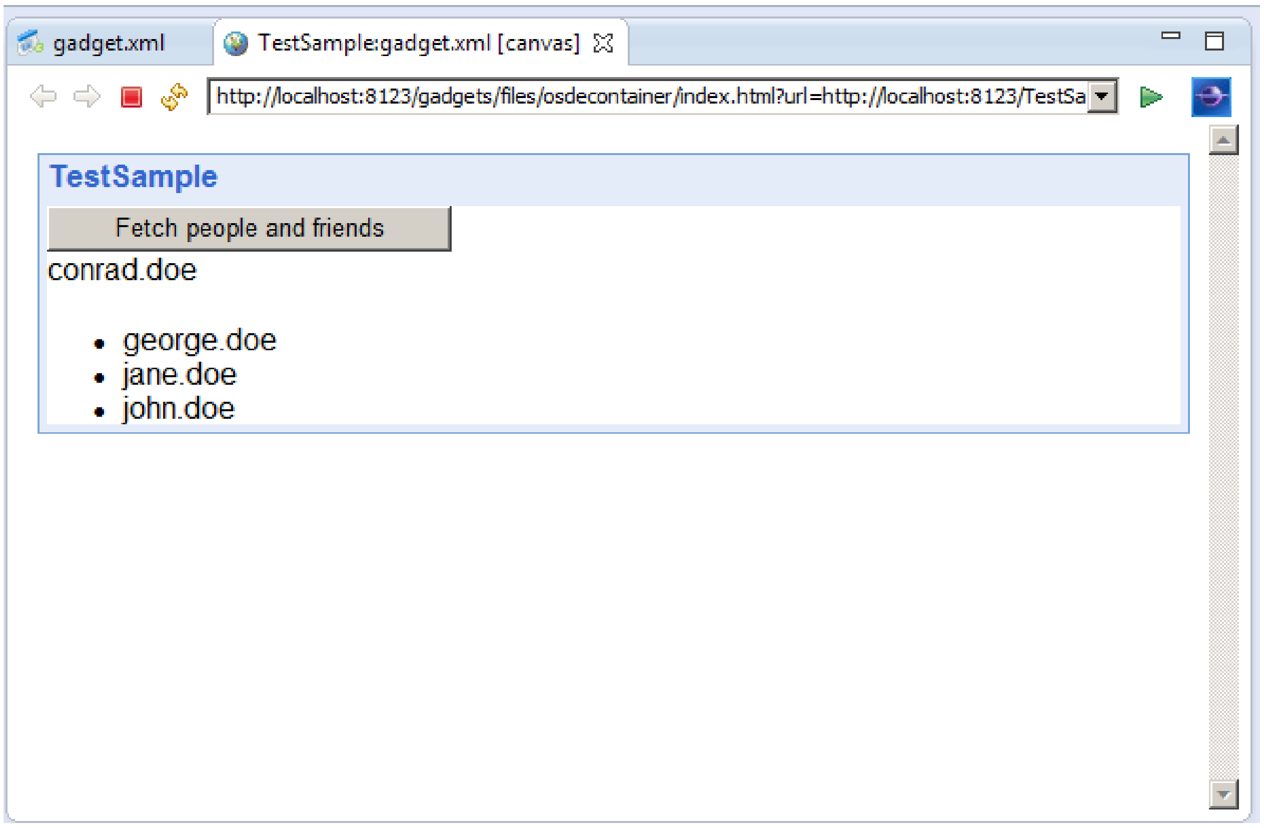
8. A browser window in eclipse shows the TestSample gadget.



```
<?xml version="1.0" encoding="UTF-8"?>
<Module>
  <ModulePrefs title="TestSample" author_email="rick_an@your.domain">
    <Require feature="opensocial-0.9" />
    <Require feature="dynamic-height" />
    <Require feature="osapi" />
/ModulePrefs>
Content type="html" view="canvas"><![CDATA[

- Fetch a person and friends code for OpenSocial 0.9 -->
v>
button onclick='fetchPeople();'>Fetch people and friends</button>
div>
 <span id='viewer'></span>
 <ul id='friends'></ul>
/div>
iv>
ript type='text/javascript'>
ction fetchPeople() {
ar batch = osapi.newBatch().
   add('viewer', osapi.people.getViewer()).
   add('friends', osapi.people.get({userId: '@viewer', groupId: '@friends'}));
atch.execute(function(result) {
```

```
ndig [Java Application] C:\Users\rick_an\Desktop\me\eclipse\jre\bin\javaw.exe (12.06.2014 18:18:57)
ain] INFO  org.mortbay.log  - Logging to org.slf4j.impl.Log4jLoggerAdapter(org.mor
ain] INFO  org.mortbay.log  - jetty-6.1.15
                               t file:/C:/Users/rick_an/Desktop/me/eclipse/pl
                                Support for /, did not find org.apache.jasper
                               Configuration  - Attempting to load Configurat
                               WebSecurityManager  - http mode - enabling Ser
                               he.EhCacheManager  - Could not find a specific
                               he.EhCacheManager  - Started EHCache named [or
                               he.EhCacheManager  - Using preconfigured EHCac
```


gadget.xml
TestSample:gadget.xml [canvas]
http://localhost:8123/gadgets/files/osdecontainer/index.html?url=http://localhost:8123/TestSa
TestSample
Fetch people and friends

conrad.doe

- george.doe
- jane.doe
- john.doe

The OSDE Plugin is a useful extension to develop and test OpenSocial gadgets. With the ability to change the workspace of Eclipse to the gadgets folder of an Apache Rave installation you can easily test your gadget in this environment.

5. Widget Overview

The goal of this chapter is to give an overview of the widgets developed and made available for release 1 of MOBiNET.

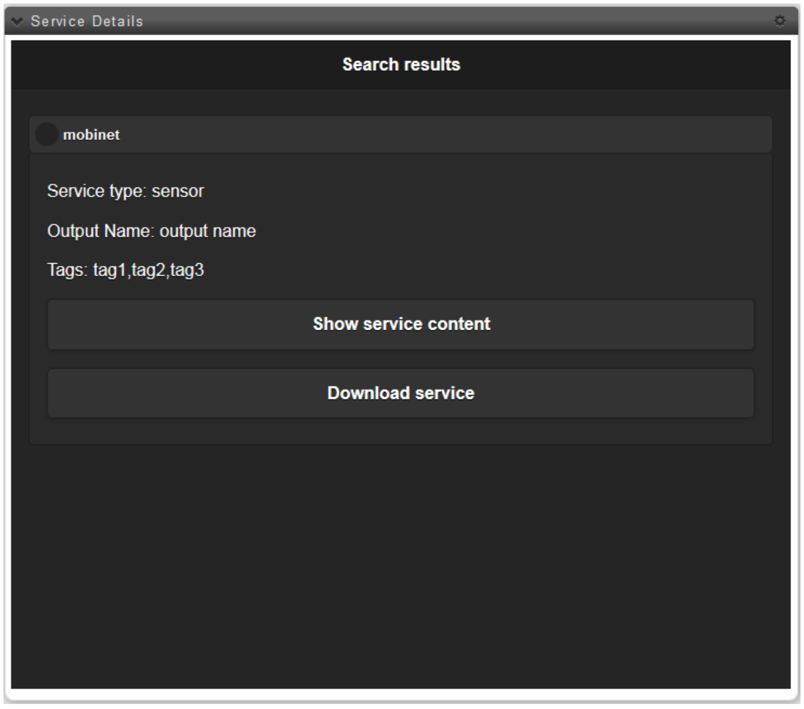
5.1 Service and App Directory Widgets

As the App Directory for release 1 of MOBiNET is based on copies of the same components that make up the Service Directory, the same goes for the widgets. For more information on the Service and App Directory please refer to the deliverable D3.2.4.1 Service Directory Concept (Release 1).

In the following the widgets developed for the Service Directory will be described briefly.

5.1.1 Service Description Details

This widget presents details of the service description obtained based on input in the Lookup and Discover service description widgets, why there is no option to give input from the user. Furthermore, the widget offers the possibility to view and download the full service description.



Service type: sensor

Output Name: output name

Tags: tag1,tag2,tag3

Figure 7 Screen capture of the Service Details Widget with an expanded service

Furthermore, in the case where multiple services match the input given in the service discover widget, all matching services will be listed, and all available for being expanded such that additional details can be seen, as shown in the figure.

5.1.2 Lookup Service Description

This widget allows the user to type in the ID of the service he is looking for, which must be known beforehand by the user. The result will be presented in the service details widget described earlier.

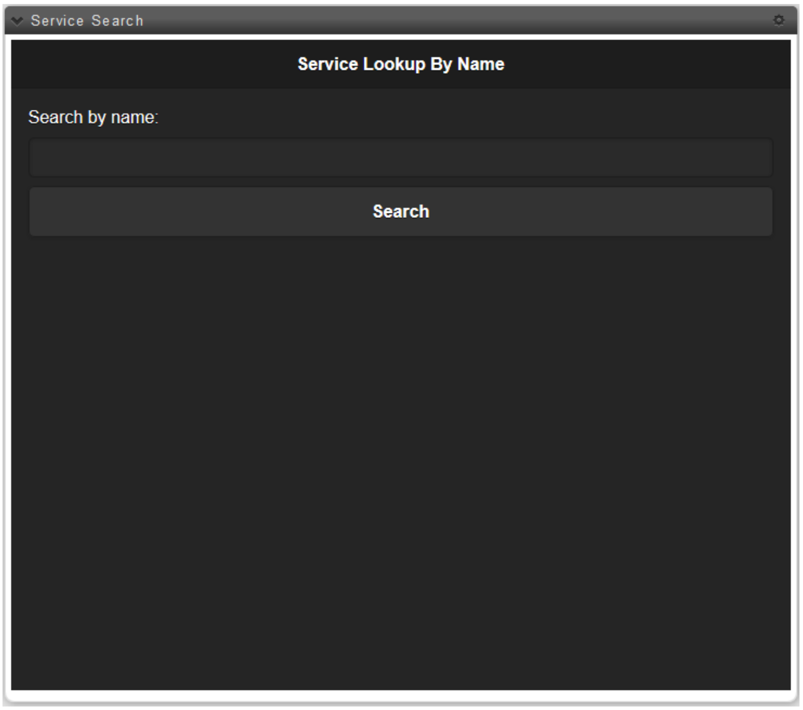


Figure 8 Screen capture of the Service Lookup Widget

Possible outcomes of a lookup action is either a matching service presented in the service details widget, or a message in the service details widget saying no service was found with the inputted service ID.

5.1.3 Discover Service Description

This widget allows the user to discover services based on service type and the service coverage area. This is done by typing in a service type, and specifying a geographical area by entering the GPS coordinates in terms of latitude and longitude of the northeast and southwest corners of an area. The services within the specified area of matching service type will now be listed in the service details widget.

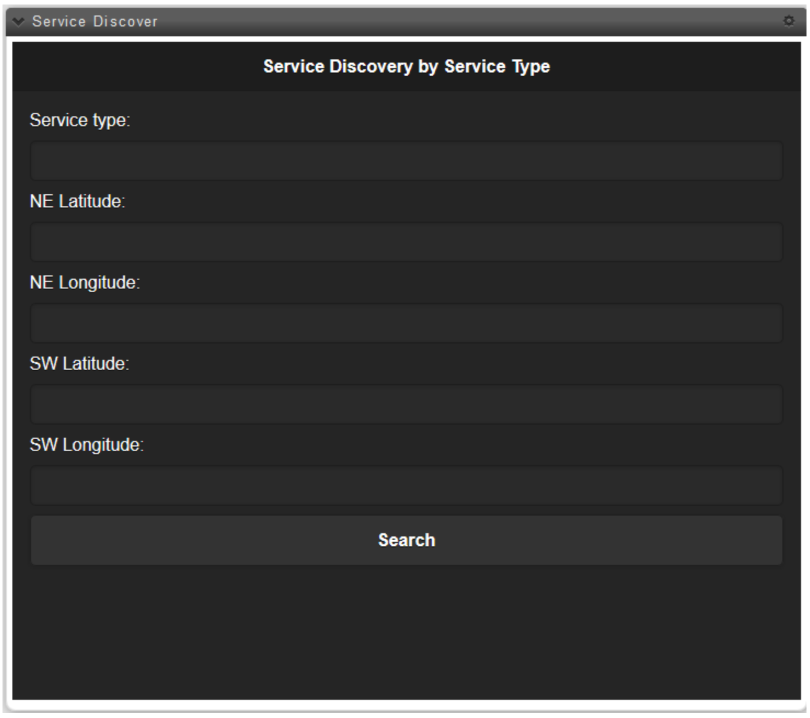
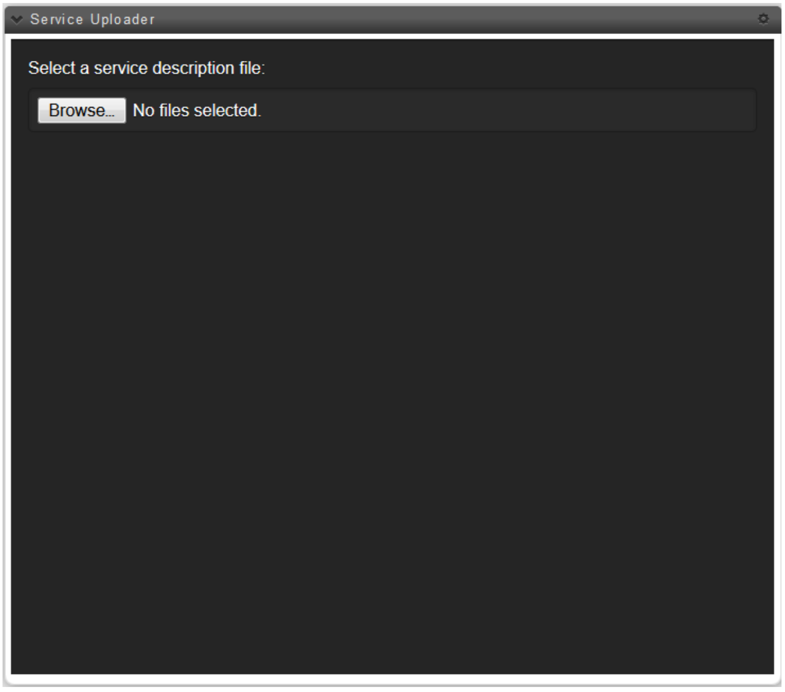


Figure 9 Screen capture of the Service Discover Widget

Possible outcomes based on the input in this widget is either a service description or a list of service descriptions in the service details widget, or a message that no matching service description was found.

5.1.4 Upload Service Description

This widget provides an interface to the user for adding service descriptions to the service directory. When the user clicks on the “Browse” button he is presented with a file browser window in which he can select a service description file. When selected, the user is prompted to confirm the upload of the selected service description. If “Ok” is selected the service description will be attempted added to the service directory, and a status message will be presented as a pop up window.



Select a service description file:

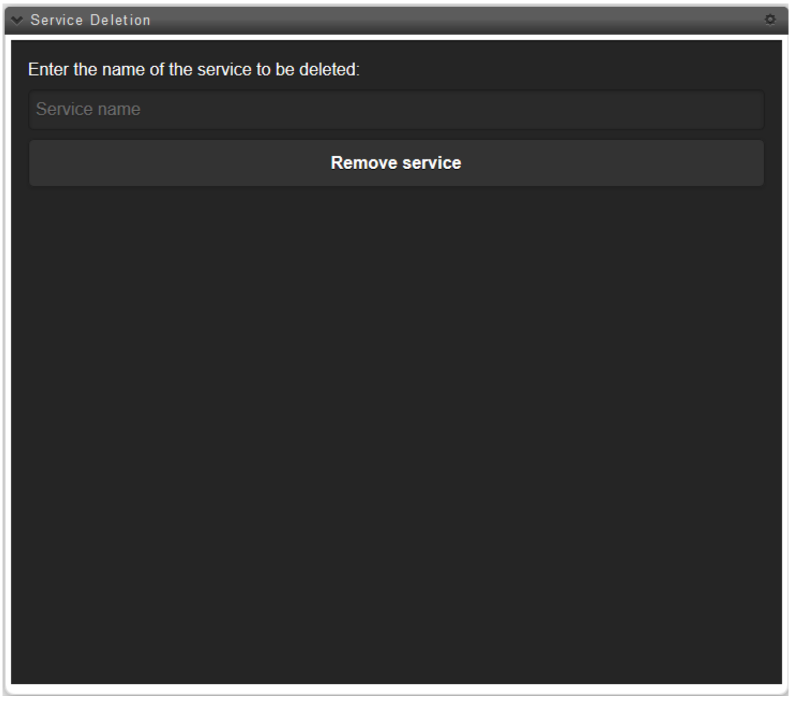
No files selected.

Figure 10 Screen capture of the Service Upload Widget

Possible outcomes of the service description upload is either successful upload of service description, or error and an appropriate error messages, such as service already in storage. Additionally if there is a structural error in the service description, the upload will be rejected.

5.1.5 Remove Service Description

This widget allows the user to remove service descriptions from the service directory based on the service ID. When the user inputs a service ID and clicks “Remove service” he is prompted to confirm this.



Enter the name of the service to be deleted:

Figure 11 Screen capture of the Service Deletion Widget

Possible outcomes of input to the widget are that the service description is removed from the service directory, or in the case where no service exists with the indicated service ID nothing is changed.

6. Summary and Outlook (NEC)

6.1 Overview of the State of the Dashboard

Currently the MOBiNET Dashboard is in a stable status allowing a problem free deployment of new OpenSocial Gadgets or W3C Widgets. As shown in chapter 5 the MOBiNET Service Directory and the MOBiNET App Directory have already deployed first versions of their OpenSocial Gadgets.

The Dashboard already integrates with the MOBiNET Identity Manager using OpenID in order to provide authentication. The Dashboard is already prepared for the release 2 integration with the Identity Manager which will provide authorization functionality. This authorization will be realised using OAuth.

6.2 Further work

For the release 2 of MOBiNET we will look into metrics to analyse the service and app usage. Hence we will integrate a Data Analytics Server component which will take care of all the data provided by the Service Directory, App Directory and the MOBiAgent. In order to get this data further integration with the Service Directory, App Directory and the MOBiAgent will be done.

Additionally we will develop new widgets to present the results of the data analysis to the service and app developers. As mentioned in chapter 6.1 the Dashboard will further integrate with the Identity Manager in order to provide authorization functionalities for the widgets and their services. The Dashboard is already prepared for the integration using OAuth.