Mastering Data-Intensive Collaboration and Decision Making

FP7 - Information and Communication Technologies

Grant Agreement no: 257184
Collaborative Project
Project start: 1 September 2010, Duration: 36 months

D5.1.2 - Standards and guidelines for development (enhanced version)

Due date of deliverable:	28 February 2013
Actual submission date:	28 February 2013
Lead Partner:	NEO
Contributing Partners:	UPM, NEO

Nature: ☒ Report ☐ Prototype ☐ Demonstrator ☐ Other

Dissemination Level:

- ☒ PU : Public
- ☐ PP : Restricted to other programme participants (including the Commission Services)
- ☐ RE : Restricted to a group specified by the consortium (including the Commission Services)
- ☐ CO : Confidential, only for members of the consortium (including the Commission Services)

Keyword List: Software quality, Integration guidelines,, Web services, Dicode Workbench, REST, SOAP

The Dicode project (dicode-project.eu) is funded by the European Commission, Information Society and Media Directorate General, under the FP7 Cooperation programme (ICT/SO 4.3: Intelligent Information Management).

The Dicode Consortium

Computer Technology Institute & Press "Diophantus" (CTI) (coordinator), Greece

University of Leeds (UOL), UK

Fraunhofer-Gesellschaft zur Foerderung der angewandten Forschung e.V. (FHG), Germany

Universidad Politécnica de Madrid (UPM), Spain

Neofonie GmbH (NEO), Germany

Image Analysis Limited (IMA), UK

Biomedical Research Foundation, Academy of Athens (BRF), Greece

Publicis Frankfurt Zweigniederlassung der PWW GmbH (PUB), Germany

Document history			
Version	Date	Status	Modifications made by
1	08-02-2013	First draft	Guillermo de la Calle Velasco (UPM) Doris Maassen (NEO)
2	12-02-2013	Second draft (sent to internal reviewers)	Guillermo de la Calle Velasco (UPM) Doris Maassen (NEO)
3	18-02-2013	Reviewers' comments incorporated (sent to SC)	Doris Maassen (NEO)
4	28-02-2013	Final (approved by SC, sent to the Project Officer)	Doris Maassen (NEO)

Deliverable manager

- Doris Maassen (NEO)

List of Contributors

- Guillermo de la Calle Velasco (UPM)
- Doris Maassen (NEO)
- Eduardo Alonso Martínez (UPM)
- Martha Eugenia Rojas Vera (UPM)

List of Evaluators

- Lydia Lau (UOL)
- Nikos Karacapilidis (CTI)

Summary

This deliverable is an enhancement of D5.1.1 which has defined the standards and guidelines for software development agreed by all Dicode partners in the beginning of the project. This document focuses on the integration guidelines which were published in an interim document titled "Guidelines for integration of services within the Dicode workbench". For Dicode's general development standards, the reader should refer to D5.1.1 which summarizes the projects guidelines on coding conventions over test-driven development and tool support.

Table of Contents

1 Introduction 5

2 Developing Services for the Dicode Workbench 6
2.1 Registration / Publication of services 8
2.2 Authentication of services 10

3 Integrating a service within the DICODE workbench 11
3.1 Sending an item 11
3.2 Receiving an item 14
3.3 Message formats 14
3.4 Preserving the state 15

4 Summary and Perspectives 15

References 16

1 Introduction

One of the first deliverables in Dicode was D5.1.1: "Standards and guidelines for development". In the beginning of the project, all partners agreed that the establishment of common guidelines for software development constitutes a fundamental necessity in collaborative projects with several partners involved. The guidelines were also designed to help new developers to get accustomed with the development standards of the Dicode consortium. This deliverable is the extended version of the former guidelines.

In the second and third year of Dicode, there has been a strong demand for guidelines for the integration of the Dicode services. This document consolidates these guidelines. Starting with a high-level description of service development in Dicode, all necessary details about service development, publishing of services and integration of services are given. The guidelines conclude with a hands-on guide about the two available integration modes: light integration and full integration.

There has been a shift in focus from the first to the second version of D5.1 from development to integration which mirrors a more general trend in software development. Currently, we notice a strong trend in the IT industry towards a diversity of programming languages and environments. Especially with the rise of new languages for Java's virtual machine like Scala and Clojure and the increasing demand for parallelism which resulted in a renaissance of functional programming, we saw a growing diversity of programming language usage in the software industry. Using the right language for a given task – or even a personally preferred tool - has become more and more acceptable.

All Dicode partners agreed on developing light-weight services which could be integrated via a REST-based interface or directly as a widget into the Dicode workbench. This pattern does not pose any restrictions on the back-end technology used for service development and supports the trend described above. In Dicode, components of the system are wrapped into services and integration is performed on a service level. Due to this agreement, the need for common coding and team organization practices could be simplified. Dicode developers did not feel the need to use common programming languages, coding guidelines or even IDEs, but adopted the coding conventions and development standards of the respective partners.

2 Developing Services for the Dicode Workbench

The Dicode Workbench has been designed and implemented as a web application. A widget-based approach has been adopted to implement and integrate services within the Dicode workbench. A web widget can be defined as a small stand-alone software application that can be embedded within a web page and executed/used by the end user. For a general description, see deliverables D5.4.1 and D5.4.2 ("Integrated Dicode Services" – initial and enhanced version, respectively).

In the Dicode workbench, widgets are distributed in three "logical" columns: two small ones on the sides and a bigger one in the middle. By default, the collaborative workspace is "maximized" and presented in the center with the rest of services "minimized" on either side. The Dicode workbench allows users to maximize any of the widgets located on the sides. When a widget is maximized, it swaps its position with the widget at the center. Users sharing a workspace are always presented with the same set of services. But users can customize their own view of the shared workspace by re-ordering the widgets. The position of the widgets is remembered from one session to the next one. Dicode workbench also allows users to search and add services that have been previously registered in the system by service providers or developers.

The only technical requirement for a service to be integrated into the workbench is that it can be loaded and displayed into an iframe. Any web application can be displayed inside an iframe. The recommendation for developers is to use state-of-the-art web technologies such as HTML5, CSS3, JavaScript or jQuery.

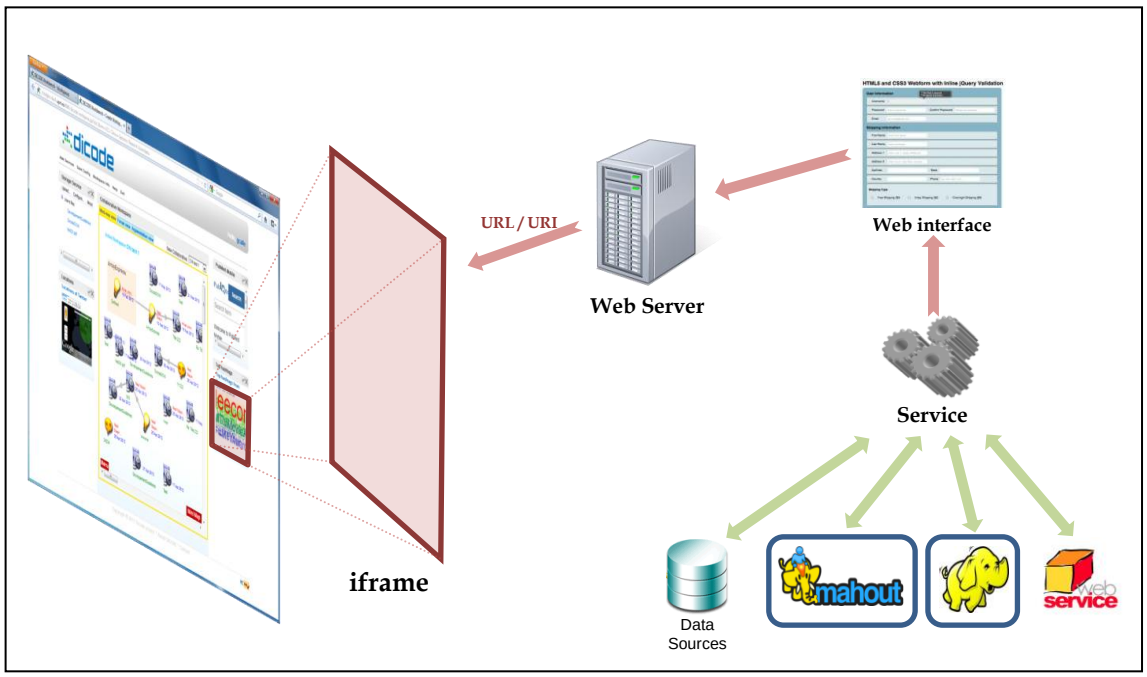

Figure 1. Structure of a service integrated within the Dicode workbench

Figure 1 depicts the structure of a service integrated within the Dicode workbench. A service performs a concrete task or set of tasks, for instance, retrieves information from a database, analyzes datasets or executes complex algorithms. The results of this task are presented through a web interface. This web interface is deployed in a web server and accessible via a

URL/URI. This URL/URI is used by the Dicode workbench to display the web interface within an iframe element.

In this way, a bidirectional communication can be established, i.e. information may flow from the service to the Dicode workbench and from the workbench to the service. For instance, communicate actions executed by the user in the workbench or data provided by the user to parameterize the service.

To integrate an application or a service in the Dicode workbench, service providers and developers should consider the following steps:

1. **Develop the service itself,** i.e. to implement the "logic" of the service. A service might be as simple as displaying a message or perform an addition, or as complicated as running complex algorithms using high performance toolkits. These services can use any technology or library, independent of the Dicode workbench.

   Developers should also take into account that the service may be invoked from another application. Thus, a public application interface should be created. We recommend to create web service interfaces based on RESTful services or WS-* (SOAP) services [1].

2. **Develop a web interface** to provide users with access to the service. Users have to be able to invoke and use the service from this web interface. In fact, this web interface acts as a wrapper for the service. Additionally, if the service needs or can be invoked with different parameters, the web interface could also provide facilities for the user to input such parameters.

   Designers and developers should consider carefully the available space devoted to widgets in the Dicode workbench. Applications for widgets are more similar to mobile applications which have a limited display area. In particular, for the Dicode workbench, widgets can have two possible states: *maximized* in the middle or *minimized* on the sides. Ideally, the web interface should adopt a liquid and elastic design [2][3].

   Depending on the type of integration required for the service, developers may also need to refer to Section *3 Integrating a service within the DICODE workbench*.

3. **Deploy the service and the web interface.** Both elements have to be accessible through the Internet via an URL/URI. Thus, they have to be deployed in a (web) server.

4. **Register or publish the service.** The Dicode workbench only can display those services that have been previously registered in the system. A registry of annotated services is maintained by the Dicode workbench. Complete publication process is explained in Section *2.1 Registration / Publication of services*.

2.1 Registration / Publication of services

To use services from within the Dicode workbench, they have to be previously registered or published in the system. The publication of services is a dynamic process, i.e. services can be published at any time by any user of the system. We tried to maintain the publication process as simple as possible, and thus, only short information about services is required. Such information is described in the following figures.

Registration of services can be done by selecting the option "Services" from the menu on the left (1) and then, clicking on the link "+ Publish a new service…", (2) as shown in Figure 2.

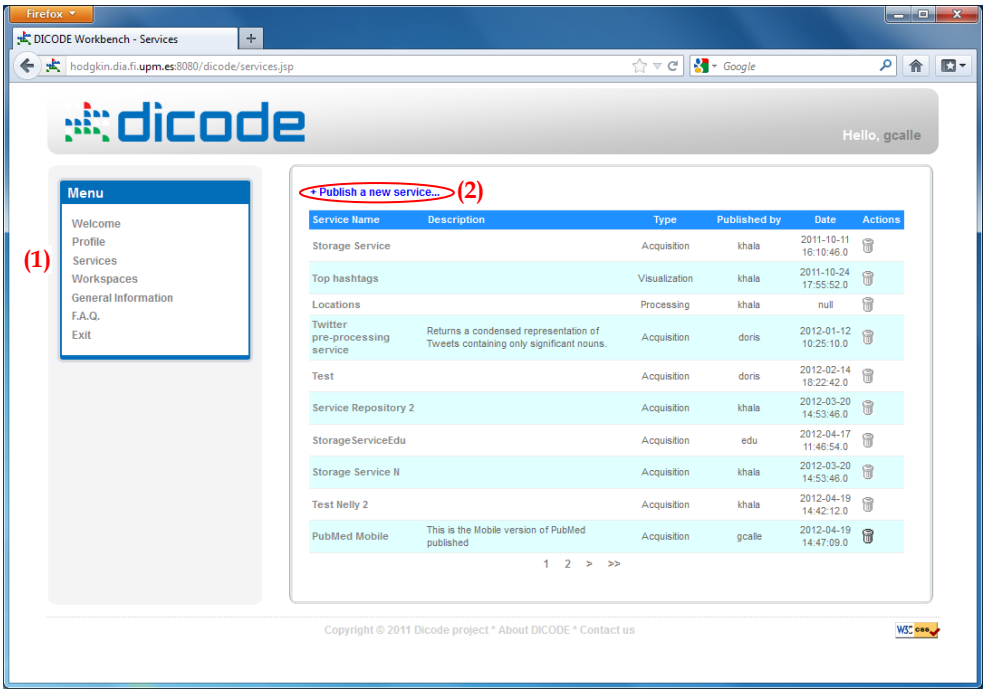

Figure 2. Registering a new Service (I)

Figure 3 shows the form to register/publish a service in the Dicode workbench.

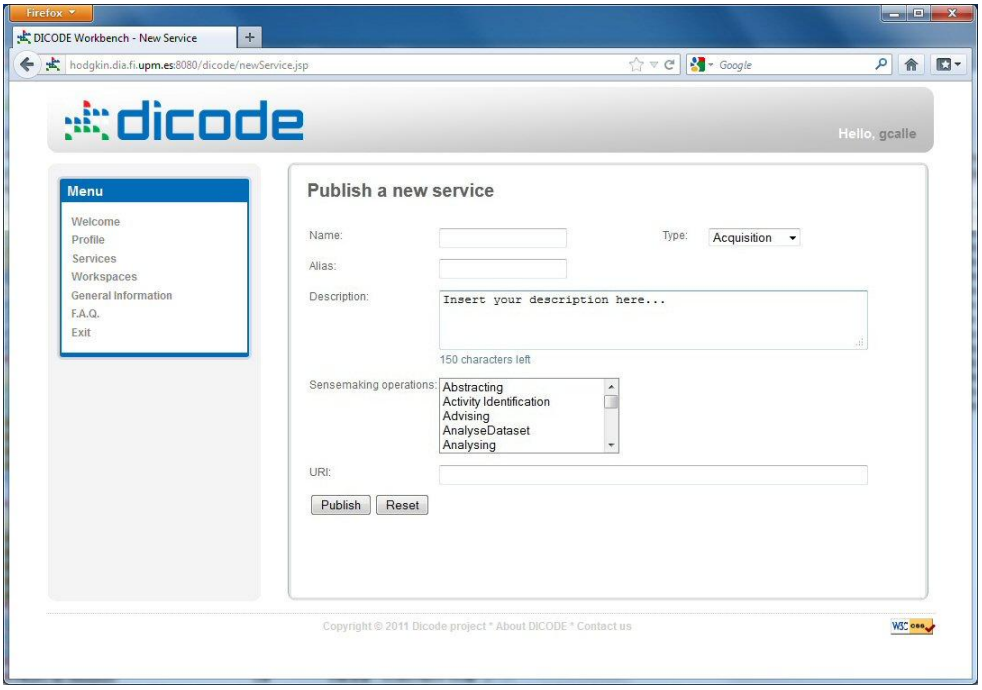

Figure 3. Registering a new Service (II)

Fields required to publish a new service within the Dicode workbench are:

1. **Name:** textual string given by the publisher or service provider to identify the service within the Dicode workbench. It is unique in the system.

2. **Alias:** short textual identifier of the service. It is used to display the name of the service on the top of the widget. It is limited to 15 characters.

3. **Type:** this attribute allows annotating the service according to the type of service. Only three values are permitted: Acquisition, Processing and Visualization.

4. **Description:** free textual description of the service. The publisher or service provider can use this field to provide information about the service; functionality, parameters required, domain, etc.

5. **Sensemaking operations:** this field is used to annotate services according to the sensemaking operations contained in the Dicode ONtology (DON). None, one or several options can be selected. Annotations are used by the Dicode workbench to facilitate user's searches when they are looking for new services to be added to their workspaces.

6. **URI:** this is the most important field regarding a service. It establishes the URI where the service is running. The Dicode workbench uses this URI as iframe source in the widget. Only services accessible via URI can be integrated within the Dicode workbench. The URI must contain the complete address (with parameters if needed) to invoke the service. Therefore, the service has to be previously deployed in a web server.

Finally, to complete the registration in the Dicode workbench, the publisher or service provider has to click on the "Publish" button. Once a service has been successfully published in the Dicode workbench, it can be added and used by users within the shared workspaces.

2.2 Authentication of services

User's authentication is carried out by the Dicode workbench when users login into the system. As agreed by the Technical Development Committee (TDC) of the Dicode project, if one service requires an extra verification due to security restrictions or policy in the organization of the service provider, invocations can be filtered by the IP address of the requester. In the case of the Dicode workbench, invocations will be done from the computer named "hodgkin.dia.fi.upm.es" with the IP address 138.100.11.177. This IP address should be added to the trusted IPs in the server or firewall of the service provider.

3 Integrating a service within the DICODE workbench

The Dicode workbench offers two different integration modes: light integration and full integration. Light integration follows a classical mashup approach, whereas full integration allows the communication between widgets (for details, see deliverable D5.4.2). In this document we focus on the practical integration aspects.

Full integration relies on the HTML5 *postMessage* mechanism which allows applications running in different windows to communicate information (plain text) across different origins and domains. In the Dicode workbench, this mechanism is triggered by drag and drop. When the Dicode workbench detects that the user wants to move an element from one widget to another, it takes the reference from the origin source and sends a message containing the reference to the target widget (Figure 4). Upon receiving the message, the target widget interprets it and performs the associated actions. In the following sections, instructions to developers for incorporating these functionalities are provided. Those instructions contain examples implemented in JavaScript [8] using the facilities of HTML5 [9]. Methods proposed can be refined by service developers by using, for instance, jQuery [10][11] or other libraries.

At the moment, interactions between widgets are triggered by users when they move elements from one widget to another. But this architecture could be extended to allow widgets/services to trigger events, to send data to other widgets/services by following a publish-subscribe design pattern [12].

3.1 Sending an item

Two actions are needed to send items in the Dicode workbench:

1) All HTML elements which can be dragged must be labeled as *draggable;*

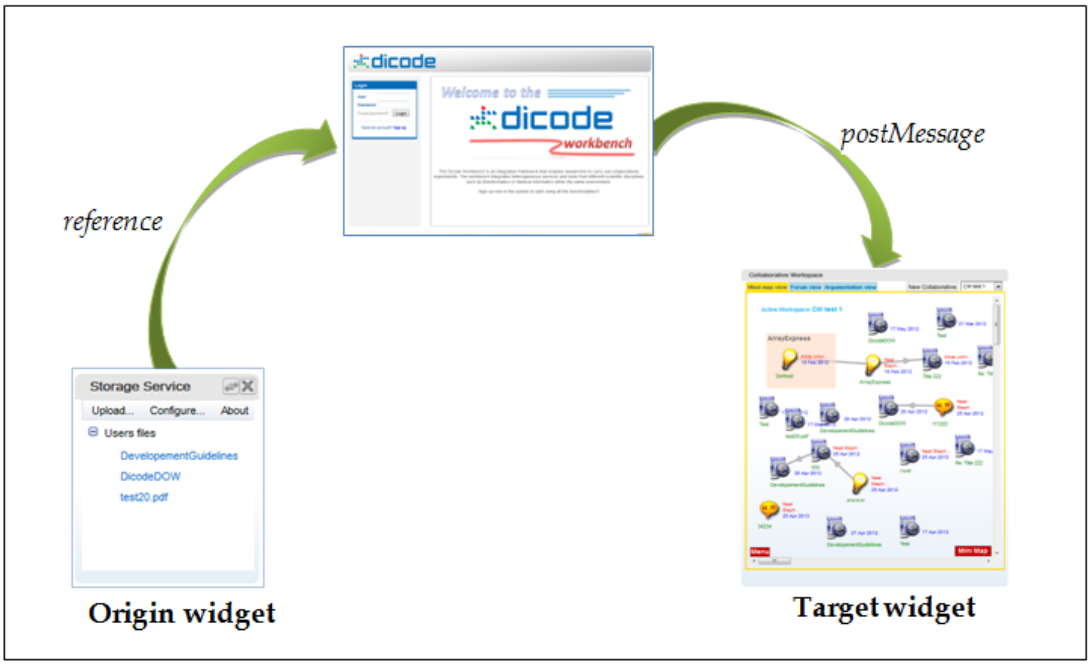

Figure 4. Communication between widgets in the Dicode workbench

2) Define the information to be sent when the item is dragged.

To label an element as *draggable*, an attribute to the tag of this element has to be added as shown in the example code in Figure 5. Thus, browsers can identify those elements as *draggable*.

```
<!doctype html>
<html>
    <head>
        <script src="js/dragDrop.js"></script>
    </head>
    <body>
        <a href="#" draggable="true" ondragstart="processDragStart(this.href, event)"> item1 </a>
        <a href="#" draggable="true" ondragstart="processDragStart(this.href, event)"> item2 </a>
        <a href="#" draggable="true" ondragstart="processDragStart(this.href, event)">  item3 </a>
    </body>
</html>
```

Figure 5. Example code to allow items to be dragged

In the code of Figure 5, a file named *dragDrop.js* containing JavaScript code is imported. This file contains the functions to handle drag behavior of an item and to receive messages. Full content of *dragDrop.js* file is shown in Figure 6.

To establish the information to be sent, add an event to the *draggable* items and define a function to process the event. Usually, this function will define the information to be exchanged between widgets. There are two options to add an event to an item:

i) Including the event in the HTML code as shown in Figure 5 o*ndragstart* is the name of the event that is triggered when users start a drag action; p*rocessDragStart* is the name of the function to process the event.
ii) Invoke the function *addEvent*, included in the code of Figure 6, for each *draggable* item. Using tools and libraries such as, for instance, cssQuery[13], the DOM of the document can be examined looking for items defined as *draggables*.

Once the *listeners* for the events are established, service developers have to codify the function/s to attend the event/s. As described above, in the Dicode project, we have adopted a message passing strategy. Thus, that function should be used to construct the message that the service wants to communicate to the other services. For the Dicode project, a preliminary set of messages have been defined (see Section *3.3 Message formats)*, but that set can be extended as needed.

```
/* dragDrop.js */

var addEvent = function(obj, evType, fn) {
                        if (obj.addEventListener) { //W3C DOM
                           obj.addEventListener(evType, fn, false);
                        } else if (obj.attachEvent) { //IE DOM
                           obj['e' + evType + fn] = fn;
                           obj[evType + fn] = function() {
                                                   obj["e" + evType + fn](self.event);
                                               };
                           obj.attachEvent("on" + evType, obj[evType + fn]);
                        }
                     };

function processDragStart(ref, e) {
                     var message;
                     message = '{"Item": {\n' +
                                 '\t"type": "File",\n' +
                                 '\t"name": "",\n' +
                                 '\t"uri": "'+ref+'",\n' +
                                 '\t"format": "",\n' +
                                 '\t"description": ""\n' +
                                 '\t}\n}';
                     if (parent.postMessage) {
                       parent.postMessage(message,"*");
                     } else {
                              alert("Your browser does not support the postMessage");
                     }
                  };

function OnMessage (event) {
        var message = event.data;
        //Check the location of the caller

       //Opera earlier than version 10
       if ('domain' in event) {
          if (event.domain != "hodgkin.dia.fi.upm.es:8080") {
             return;
          }
       }

       // Firefox, Safari, Google Chrome, Internet Explorer from version 8 and Opera from version 10
       if ('origin' in event) {
          if (event.origin != "http://hodgkin.dia.fi.upm.es:8080") {
             return;
          }
       }
       // TODO Treatment of the received message
       alert("RECEIVED: "+message);
   };

onload = function () {
                       addEvent(window, "message", OnMessage);
                     };
```

Figure 6. Complete code of JavaScript file *dragDrop.js*

In Figure 6, an example of such function is provided. In this case, one message is created following the message structure adopted. After message is created, it is sent to the parent window, i.e. the Dicode workbench.

3.2 Receiving an item

To receive messages in an application, two actions have to be carried out:

1) Create a *listener* to receive messages in the application/service. An example of how to create such a *listener* is shown at the end of Figure 6. This *listener* will be associated to the window/iframe where the application is running.

2) Define and codify the treatment of the information received into the message. In the example of Figure 6, the function *OnMessage* has been defined for such purpose. In this case, the function *OnMessage* checks the origin of the message to prevent from unauthorized uses, and then the content of the message is shown in a popup window. Treatment of messages can be as simple as presented, but it can be as complex as service developers need.

3.3 Message formats

Message passing approach requires that both sender and receiver agree on a common format for exchanging information. For Dicode project, we have adopted a message format based on JSON [14][15]. A preliminary set of basic message formats has been defined to be used by the applications/services within the Dicode workbench (see figure below - note that parts highlighted in yellow have to be completed by the sender with the proper information). This set of formats is not a closed list; it could be easily expanded with more types of messages.

Type of item	Message format

File	{Item: {
	type: File,
	name: name,
	uri: uri,
	format: format,
	description: description
	}
	}

	{Item: {
	type: Image,
	name: name,
	uri: uri,
	description: description
	}
	}

Text	{Item: {
	type: Text,
	content: content
	}
	}

	{Item: {
	type: Link,
	uri: uri
	}
	}

3.4 Preserving the state

When a user swaps a "maximized" service with a "minimixed" service, both services need to maintain their current views respectively to ensure continuous usage during a session. To facilitate this task, Dicode workbench sends to services some extra information as HTTP parameters during the invocation call. Services are available through an URL and they are loaded in an *iframe* element in the workbench. Two extra parameters are sent to services using the query string:

- **expid –** it is an integer number identifying the current workspace
- **usrid –** it is an integer number identifying the current user. This number is unique for each user in the Dicode workbench.

Example of invocation:

*http://hodgkin.dia.fi.upm.es:8080/StorageService?width=279&height=215&color=0066bb &**expid**=3&srvid=1&**usrid**=2"></iframe>*

It is up to services to process appropriately those parameters to carry out the appropriate action to preserve the state of services between invocations.

4 Summary and Perspectives

This document mainly focusses on the integration of Dicode services. Besides the current members of the Dicode development teams, this deliverable also serves the needs of new staff joining the project and developers from outside the consortium.

The general "Standards and guidelines for development" defined in the first version of this deliverable can be found in the Dicode wiki at https://wiki.dicode-project.eu/display/DIC/D5.1+-+Standards+and+guidelines+for+development. The content of this document is up to date and will be updated on the wiki if changes occur.

References

[1] Cesare Pautasso , Olaf Zimmermann , Frank Leymann, Restful web services vs. "big"' web services: making the right architectural decision, Proceeding of the 17th international conference on World Wide Web, April 21-25, 2008, Beijing, China [doi>10.1145/1367497.1367606]

[2] Cederhlm, D.: Bulletproof Web Design: Improving flexibility and protecting against worst-case scenarios with XHTML and CSS. New Riders, Berkely, USA (2006)

[3] Liquid layout. [http://webdesign.about.com/od/layout/g/bldefliquidlyot.htm]

[4] iGoogle. [http://www.google.com/ig]

[5] Message Passing Interface. [http://en.wikipedia.org/wiki/Message_Passing_Interface]

[6] HTML5 Web Messaging. [http://dev.w3.org/html5/postmsg/]

[7] Cross-document messaging. [http://en.wikipedia.org/wiki/Cross-document_messaging]

[8] JavaScript – dataTransfer object. [http://help.dottoro.com/ljmpcqdb.php]

[9] HTML5. [http://dev.w3.org/html5/spec/]

[10] jQuery. [http://jquery.com/]

[11] Ben Alman – jQuery postMessage. [http://benalman.com/projects/jquery-postmessage-plugin/]

[12] Publish-subscribe pattern. [http://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern]

[13] cssQuery. [http://dean.edwards.name/my/cssQuery/]

[14] JSON. [http://www.json.org/]

[15] JSON. [http://en.wikipedia.org/wiki/JSON]