The Dicode project (http://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

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Status</th>
<th>Modifications made by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10-10-2011</td>
<td>Initial outline</td>
<td>Guillermo de la Calle (UPM)</td>
</tr>
<tr>
<td>2</td>
<td>04-11-2011</td>
<td>First draft</td>
<td>Guillermo de la Calle (UPM), Víctor Maojo (UPM), Miguel García-Remesal (UPM), Nelida Nkumu-Mbombio Ada (UPM), Eduardo Alonso (UPM)</td>
</tr>
<tr>
<td>3</td>
<td>19-11-2011</td>
<td>Second draft (sent to internal reviewers)</td>
<td>Guillermo de la Calle (UPM), Nelida Nkumu-Mbombio Ada (UPM), Eduardo Alonso (UPM), Manolis Tzagarakis (CTI), Spyros Christodoulou (CTI), Georgia Tsiliki (BRF)</td>
</tr>
<tr>
<td>4</td>
<td>26-11-2011</td>
<td>Internal reviewers’ comments incorporated, sent to SC</td>
<td>Guillermo de la Calle (UPM)</td>
</tr>
<tr>
<td>5</td>
<td>30-11-2011</td>
<td>Final version (approved by SC, sent to the Project Officer)</td>
<td>Guillermo de la Calle (UPM)</td>
</tr>
</tbody>
</table>

**Deliverable manager**
- Guillermo de la Calle, UPM

**List of Contributors**
- Guillermo de la Calle, UPM
- Víctor Maojo, UPM
- Miguel García-Remesal, UPM
- Nelida Nkumu-Mbombio Ada, UPM
- Eduardo Alonso, UPM
- Manolis Tzagarakis, CTI
- Spyros Christodoulou, CTI
- Nikos Karacapilidis, CTI
- Georgia Tsiliki, BRF

**List of Evaluators**
- Dhaval Thakker, UOL
- Max Jakob, NEO

**Summary**

This deliverable describes the initial version of the suite of integrated services developed within the Dicode project framework. It reports on major development efforts carried out within Tasks 5.3 (Dicode user interfaces) and 5.4 (Meaningful technical integration of Dicode services). A preliminary version of the Dicode workbench is presented, describing its internal design and all the functionalities provided to the users. In addition, this deliverable provides a detailed view of the strategy selected by the consortium to integrate services developed in WP3 and WP4, as well as external services. First versions of the already integrated services are also presented. Finally, some usage examples - focusing on the project’s specific use cases - are described in detail, illustrating the foreseen workspaces that end-users will work with in the context of the Dicode project.
Table of Contents

1 Introduction....................................................................................................................... 6

2 Dicode Workbench........................................................................................................... 7
  2.1 General description ....................................................................................................... 7
  2.2 Site map ....................................................................................................................... 7
  2.3 User roles .................................................................................................................... 9

3 Integration of services .................................................................................................... 11
  3.1 Dicode integration approach ....................................................................................... 11
    3.1.1 Widgets .................................................................................................................. 11
    3.1.2 Dicode Registry of Services .................................................................................. 13
    3.1.3 Single sign-on strategy ......................................................................................... 14
  3.2 Suite of services integrated .......................................................................................... 16
    3.2.1 Workbench management services ......................................................................... 16
      3.2.1.1 User management service .............................................................................. 17
      3.2.1.2 Service management service ......................................................................... 26
      3.2.1.3 Workspace management service ................................................................... 34
    3.2.2 Collaboration support services ............................................................................. 44
      3.2.2.1 Description .................................................................................................... 44
      3.2.2.2 Interface overview ....................................................................................... 45
    3.2.3 Storage service .................................................................................................... 48
      3.2.3.1 Description .................................................................................................... 49
      3.2.3.2 Interface overview ....................................................................................... 52
      3.2.3.3 Internal design .............................................................................................. 54

4 Dicode workbench user manual ..................................................................................... 59
  4.1 Main page ................................................................................................................... 59
    4.1.1 Create an account ................................................................................................. 59
    4.1.2 Remember password ......................................................................................... 62
  4.2 Login into the system .................................................................................................. 63
    4.2.1 Welcome page ....................................................................................................... 64
    4.2.2 Profile section ....................................................................................................... 64
    4.2.3 Services ................................................................................................................ 67
      4.2.3.1 Publish a new service .................................................................................... 68
    4.2.4 Workspaces ........................................................................................................... 70
      4.2.4.1 Create a new workspace ............................................................................... 71
      4.2.4.2 Adding new services to the workspace ......................................................... 73
    4.2.5 General Information ............................................................................................ 74
    4.2.6 Frequently Asked Questions (FAQs) .................................................................... 75

5 Usage examples .............................................................................................................. 77
  5.1 Use case 1 ................................................................................................................... 77
    5.1.1 Scenario 1 (Linking genomic data to proteomic/gene ontology (GO) data) ....... 77
    5.1.2 Envisioned workspace for U1_Scenario 1 ............................................................ 78
  5.2 Use case 2 ................................................................................................................... 79
    5.2.1 Scenario 1 ............................................................................................................. 79
5.2.2 Envisioned workspace for UC2_Scenario 1 ................................................................. 81
5.3 Use case 3 .................................................................................................................. 82
  5.3.1 Scenario 1 ........................................................................................................... 82
  5.3.2 Envisioned workspace for UC3_Scenario 1 ...................................................... 82

6 Conclusions and Future Work ...................................................................................... 84

References ...................................................................................................................... 85
1 Introduction

The two main objectives of WP5 within the Dicode project are: (i) to provide a seamless integration and interoperability of the services developed by the consortium, and (ii) to develop innovative work methodologies in data-intensive collaboration. The current deliverable focuses on the first objective (the deliverable D5.5 shall cover the innovative work methodologies in data-intensive collaboration).

This deliverable presents the work carried out in Tasks 5.3 and 5.4. Task 5.3 deals with the development of user interfaces. For the development of the user interfaces, an iterative user-centered methodology is being applied. End-users are participating from the beginning in the design of the interface, providing inputs and suggestions to continuously improve the system. The objective of this task is to create a flexible software architecture, customizable for different scenarios (use cases), to allow the integration of already existing services with new services specifically developed for Dicode project. As an added value, all the implementation will be using existing open source frameworks, toolkits and libraries to ensure their reusability, scalability and expandability.

Task 5.4 aims at developing an integration framework to facilitate the coordination and interoperability of services in the Dicode project. This integration has to be carried out from both conceptual and technical point of views. That means not only to provide users with an integrated environment to use different services, but also to create a complete integration framework where services can interoperate. In Dicode, the integration framework has been designed considering the functional requirements established in WP2, oriented to provide support and access to the suite of services developed for the project’s use cases.

Deliverable 5.4.1 is envisioned as a reference point useful for both developers and end-users. Section 2 introduces the Dicode workbench, the integration platform for all Dicode data analysis and collaboration services. The workbench has been conceived as a web application; its current site map is presented together with a general description of the application. Also, an overview of the identified user roles for this application is provided. Section 3 presents the integration framework designed for the Dicode project. First, the Dicode integration approach is explained, detailing the integration strategy adopted by the consortium. Then, the set of services that has been integrated within the Dicode workbench are presented. Section 4 contains the user manual of the Dicode workbench. This manual provides the necessary help to the end-users of the Dicode project and contains detailed information about the Dicode workbench application. Section 5 depicts some usage examples of the Dicode workbench centered in the three use cases of the project. Related scenarios are presented together with the envisioned workspace that will be available for the end-users to facilitate their work. Finally, Section 6 draws conclusions regarding the initial version of the development work described in this deliverable.
2 Dicode Workbench

2.1 General description

One objective of WP5 is to provide users with a common graphical interface to access and use heterogeneous services. The Dicode workbench is the integration platform for all Dicode data analysis and collaboration services. It has been designed as a web application. After analyzing the different possibilities available, the Dicode consortium decided to adopt a web-based solution for the following reasons:

- Web applications are world-wide accessible just using a computer with a web browser and Internet connection.
- Users do not need to install any extra application on their computer. Therefore, no problem with virus or malware applications will arise.
- Web applications are platform and operating system independent.
- Security problems are avoided since most web applications are firewall friendly, using common ports that usually are not filtered by security policies of organizations. That way, applications availability is ensured.
- Workspaces and applications can be easily and directly shared by different users.

All these points reflect specific necessities of the Dicode project that web-based systems provide, and for that reason, this approach was selected. As most web applications, the Dicode workbench has two different views: public and private. The public view is accessible by any user with Internet connection. It contains general information of the workbench and allows users to register in the system. The private view is only available for registered users.

Since web applications execute within web browsers, it is crucial that they can properly run on the most popular ones. In the Dicode project, we have paid much attention to this issue to ensure that users have all functionalities available independently from the operating system or web browser used. Concretely, we have considered five of the most popular web browsers: Internet Explorer, Mozilla Firefox, Google Chrome, Safari and Opera. At the moment, most functionalities work properly in these browsers, although some errors are still present and are being fixed. The first functional prototype of Dicode workbench has been implemented using Java technologies, i.e. JavaServer Pages (JSPs) and Servlets (Hall, 2011). It has been deployed and is publicly available for testing purposes at http://hodgkin.dia.fi.upm.es:8080/dicode. Any user is allowed to create new accounts.

2.2 Site map

Figure 2.1 shows the current site map of the Dicode workbench. As stated above, all pages have been implemented using JavaServer Pages (JSP) technology (Hall, 2011). Arrows coming from a web page denote the different options that users have available on a page. White rectangles specify those options. In this graphic, a special structure has been defined called “Confirmation/Error Schema” (C/E Schema). This schema represents the possible results coming from some pages. When users complete an action, this action may finish with success or with an error. C/E Schema models these situations, showing a confirmation page when actions finish successfully, and an error page otherwise.
Figure 2.1: Dicode workbench site map
The initial page (index.jsp) shows a welcome message and allows users to login into the Dicode workbench. Departing from this page, the application web pages of the Dicode workbench can be grouped into three main categories according to their functionalities:

1. **Unregistered user pages.** This category is composed by those pages that users can access without register themselves into the system. There are three pages:
   a. *Index.jsp* – the initial page as described above.
   b. *Registration.jsp* – this page allows users to create a new user in the Dicode workbench.
   c. *Remember.jsp* – this page allows users to recover their password to access the system whenever they forgot it.

2. **Registered user pages.** This category consists of the options that users have available after a successful login process. Those available options are displayed inside a menu box on the upper left side of the page.
   a. *Welcome.jsp* – this is the initial page when users login into the system. General information about the workbench and the latest news are displayed.
   b. *Profile.jsp* – this page allows users to display and update their personal information in the system.
   c. *Info.jsp* – this page shows general information about the Dicode project: main objectives, partners, contact information, etc.
   d. *Services.jsp* – this page presents information about the services that service providers have published in the Dicode workbench.
   e. *Workspaces.jsp* – this page lists the workspaces that users have created within the Dicode workbench.
   f. *Exit* – this option allows users to logout of the system.

3. **Workspace page.** This page constitutes the main working page for end-users. It allows users to use the services integrated in the system, personalize the workspace to adapt it to their needs, and collaborate with other users to carry out different tasks together. In fact, this page belongs to the previous category, but due to its relevance in the workbench, we decided to consider it as a separate category.

Detailed information about the contents of each page and the options available for users are provided in Section 4.

### 2.3 User roles

The Dicode workbench will be used by different types of users. At the moment, three main user roles have been identified:

- **Role 1: service provider.** The Dicode workbench provides an integrated working area to facilitate users to use and execute different services. Before using any service, such services have to be registered into the system. Service providers are the ones responsible to carry out this registration process.
• **Role 2: end-user.** This role constitutes the main role in the Dicode workbench. It represents users that use the collaboration and data mining services, i.e. researches, analysts, radiographers, etc. Since the collaborative workspaces are customizable, end-users will not only use the services but they will also configure the environment, adding and deleting services according to their specific needs. This personalization process will be detailed later.

• **Role 3: external user.** This role represents users that want to use the Dicode services outside the Dicode workbench, for instance, to integrate some services in an external environment.

Currently, the Dicode workbench has implemented Roles 1 and 2. In fact, no distinctions are made between them, i.e. all users can publish services as service providers and, at the same time, participate in several workspaces as end-users. Regarding the third role, it is not implemented at the moment.
3 Integration of services

3.1 Dicode integration approach

As mentioned above, the Dicode workbench has been implemented as a web application aimed to integrate heterogeneous services, from data mining services to collaboration services. The workbench should present the different services under a common graphic interface and allow users to use them together.

The integration has to be done at two levels: (i) at user interface (UI) level, and (ii) at operational level. Integration at UI level deals with the visualization of services under a unique environment or application. After reviewing the state-of-the-art concerning graphical user interfaces (GUIs) on web applications, the consortium decided to adopt a widget-based approach. Widgets allow users to personalize the workbench as needed. More detailed information about this topic is given in next section. On the other hand, integration at operational level is aimed to ensure the communication and exchange of data between different services. On this regard, we decided to design and implement a registry of services to store metadata about services (see Section 3.1.2 for details). To complement the registry of services, a Single Sign-On (SSO) approach has been adopted. This approach facilitates credential delegations between services. Users only have to login the system once to use all the existing services of the workbench.

In this section, the abovementioned strategies are explained in detail, introducing the general concepts and how each approach is being used in the Dicode project.

3.1.1 Widgets

A software widget is a generic type of software application comprising portable code intended for one or more different software platforms (Software widget, 2011). Since the 1980s, when the term “widget” appeared (Swick and Ackerman, 1988), different types of widgets have come up including GUI widgets, disclosure widgets, desktop widgets, widget applications and web widgets.

Web widgets in particular are software applications/widgets designed for the web and may be embedded and executed within a web page accessed by the end user (Web widget, 2011). They are stand-alone applications and one of their most important features is that their content is not controlled by the widget host. Although the widget content is read-only by the widget host, the end user may interact with the widget, as long as such functionality is provided. On the other hand, the host is able to modify the way the widget appears on the host page such as the location or size, and even to establish some initialization parameters in the URI.

A number of Web widget toolkits are nowadays available aiming at providing sets of widgets to be used when designing applications with graphical user interfaces. Some of the most popular categories of widget toolkits include low level (integrated in the operating system or on top of it as a separate application) and high level widget toolkits (operating system specific or cross platform). Dojo Toolkit (Russell, 2008), jQuery (Lindley, 2010), Abstract Window Toolkit (Zukowski, 1977), Google Web Toolkit (GWT) (Hanson and Tacy,
2007), YUI\(^1\), Sencha (formerly Ext JS\(^2\), TIBCO (Brown, 2011), DHTMLX\(^3\) or Swing (Eckstein et al., 1998) are popular widget toolkits used to create web widgets.

**Widgets in Dicode**

In the context of the Dicode project, a number of services will be developed that include scalable data mining, collaboration support and decision making support services. Most of these services (at least services that users interact with them directly) must be provided to end-users in an easy-to-use way, so as to facilitate their proper use.

These services must be provided to end-users in an easy-to-use way, so as to facilitate their proper use. Towards achieving this, a widget based approach has been adopted to deliver the Dicode services to end-users, which constitute the technical framework for development of the Dicode workbench. It will be developed in order to host and integrate the diverse services that each partner of the project will develop. Widgets of the Dicode workbench will integrate and provide access to all of the services implemented in the context of the project.

The initial version of the Dicode workbench features the widget based integration of the collaboration and decision making support services. The widget at the center of the Figure 3.1 shows an integration of different collaboration services in the Dicode workbench. As shown, other widgets can provide access to other Dicode services such as data mining services.

---


\(^3\) DHTMLX. [http://dhtmlx.com/](http://dhtmlx.com/)
In order to select the appropriate framework and build the Dicode workbench, different existing widget toolkits has been analyzed. The widget toolkit selected was the Google Web Toolkit (GWT) (Hanson and Tacy, 2007), based on Java and providing a set of core Java APIs and Web widgets. By using GWT, the widget developer may create widgets by writing source code in Java (or by using the GWT Designer provided); the source code produced is, then, compiled to browser “executable” code (JavaScript, HTML, Ajax code) running across all major browsers, including mobile browsers. This is the main benefit of the GWT and the main reason why it was selected. GWT produces different versions of “executable” code, one for each browser, while the developer has to write only one version of code in Java. From a browser’s “point of view”, each GWT widget (“frame” in GWT terms) is transformed, after the Java code has been compiled, to an HTML “iframe” tag (HTML IFRAME element, 2011).

3.1.2 Dicode Registry of Services

In order to ensure access and facilitate location of services, in the Dicode project we have implemented a Registry of Services. The idea is that this registry stores information (metadata) related to services, such as service name, service provider, location or functionality. Dicode Registry of Services (DRS) has to deal with:

- **Available services.** To be used within the Dicode workbench, all services must be registered in the DRS. Lifecycle of services in the Dicode workbench is envisaged as follows: (i) service providers develop a new service, (ii) service providers publish the service, (iii) end-users look for services according to their needs, and (iv) end-users add new services to their workspaces to be collaboratively used by the participants in that workspace.

- **Semantics.** We plan to semantically annotate the services with concepts contained in the Dicode ONtology (DON). Such annotations will be stored and managed by DRS. During the publication of services, service providers are asked to annotate manually the new services with concepts retrieved from DON. Such concepts are presented to service providers in the same publication web form, allowing them to select all concepts needed. Services can be annotated according to their functionality, domain, inputs and outputs.

- **Monitoring information.** This information could be useful in the future to establish some metrics about resources (times used, comments of users, successful attempts, average/min/max times of execution, reliability, etc.).

The main objective for storing all this information is to provide support to:

- **Data Mining and Collaboration & Decision support services.**

- **Recommendation services,** that use the information of DRS to suggest similar services to users, or even to provide fault tolerance in the future.

- **Searching services,** to facilitate the location and use of services.
Figure 3.2: Representation of the Dicode Registry of Services

Figure 3.2 presents the logical view of DRS. As shown, DRS stores the information about the services together with some extra information (metadata) retrieved from DON. In this figure, concepts from DON are represented as green circles. One service can be annotated with none, one or several concepts.

To facilitate users to move or to exchange data from one service to another, in Dicode we are working on the definition and implementation of “drag & drop” functions between widgets. That way, when users need to pass some data between services, s/he will only need to select the element by clicking on it and drag it to the destination service. Dicode workbench, using the information stored in the DRS, will perform the actions needed to carry out the movement and to establish a workflow.

3.1.3 Single sign-on strategy

Single Sign-On (SSO) refers to a user authentication process allowing a user to provide credentials for one (central) application and, once authenticated, get access to a number of other applications (Single Sign-on, 2011). Benefits of SSO include improved user and developer productivity and simplified administration but, on the other hand, a number of problems emerge in systems using SSO; it’s a rather difficult and time consuming approach to be implemented, especially when dealing with applications that have been already implemented. Apart from this drawback, applications using the SSO are vulnerable to attacks, since an intruder takes control of the central application providing the authentication, he may be authorized to access all the other decentralized applications. Moreover, the central authentication system has to be always available in order to avoid denial of services of all the other applications. A number of common SSO approaches are available (List of Single Sign-on Implementations, 2011) including among others configurations based on Kerberos, Smart Card and OTP token approaches.

SSO in Dicode

As already stated, in the context of Dicode, a number of services will be integrated in the Dicode workbench. Considering that a number of the Dicode services to be integrated
require input of user credentials, we plan to use a SSO approach in order to avoid multiple input of user data in all the services (widgets) available via the Dicode workbench.

Towards achieving this, a number of REST-based operations have been implemented to facilitate SSO between the Dicode workbench and the collaboration services. A user who wants to get access to the Dicode workbench provides his credentials to the Dicode workbench and once authenticated, the workbench propagates such credentials to the collaboration services by using appropriate REST calls. Figure 3.3 illustrates this process. The REST-based services allow the workbench to perform operations such as “Login”, “Logout”, “Register” new users and “Update Profile” of existing users in the context of the collaboration services. Depending on the progress of the Dicode workbench, additional REST-based services will be provided in the future version of the workbench. Dicode services that require user authentication will use a similar schema when integrated in the workbench, in the near future.

![Figure 3.3: Using SSO to pass parameters to collaboration services](image)

The necessary attributes used for the REST-based services (login, password, profile data, etc.) are encrypted, as a security precaution, using the AES (Advanced Encryption Standard, also known as Rijndael) (Westlund, 2002). AES provides symmetric key encryption and, in the case of Dicode, uses a key and a block size of 128 bits each to encrypt (in the level of the workbench) and decrypt (in the context of the collaboration services) the necessary data. A number of libraries provide implementations of the AES in diverse environments; for data encryption in Dicode, the Bouncy Castle Crypto Library\(^4\) was used, a library implemented in Java, well suited to the source code of Dicode workbench (AWT), also written in Java.

Table 3.1 depicts some examples of the REST calls used to perform actions concerning the collaboration services. Once the user has logged in into the Dicode workbench, SSO is used to perform authentication operations in the context of the collaboration services. As a further security measure, a timestamp variable is used to secure that each REST call remains valid for only 10 minutes since its firing from the workbench. After this period of time, the server providing the collaboration services denies the REST service execution.

\(^4\) [http://www.bouncycastle.org/](http://www.bouncycastle.org/)

<table>
<thead>
<tr>
<th>Service</th>
<th>Original Parameters (from Dicode workbench)</th>
<th>REST Calls with Encrypted Parameters (passed to service provider)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>user=ken</td>
<td><a href="http://server/user/?op=login&amp;hashData=T8Nfp%2FU%2F1Lz%2Byegfg2rAE4A6eITMueQ%2BD%2BkvHFFwp8kqnXQkzD9xxiKJ2jhv8n">http://server/user/?op=login&amp;hashData=T8Nfp%2FU%2F1Lz%2Byegfg2rAE4A6eITMueQ%2BD%2BkvHFFwp8kqnXQkzD9xxiKJ2jhv8n</a></td>
</tr>
<tr>
<td>Logout</td>
<td>user=ken</td>
<td><a href="http://server/user/?op=logout&amp;hashData=5TiGmH6KqzynxgikO47cw3idyjimir90D362x94%2B9jhIUo6Zjxpts8VQwvhD3IC">http://server/user/?op=logout&amp;hashData=5TiGmH6KqzynxgikO47cw3idyjimir90D362x94%2B9jhIUo6Zjxpts8VQwvhD3IC</a></td>
</tr>
<tr>
<td>Register</td>
<td>user=ken</td>
<td><a href="http://server/user/?op=register&amp;hashData=T8Nfp%2FU%2F1Lz%2Byegfg2rAE7f%2BPzQfiamm%2FY%2FC451XA6E75jVFA1jD1mrPxU%2Br5MR1T6Ov5a0nITz%0D%0AMenbqv%2BxnD90mn%2BnZjfljG">http://server/user/?op=register&amp;hashData=T8Nfp%2FU%2F1Lz%2Byegfg2rAE7f%2BPzQfiamm%2FY%2FC451XA6E75jVFA1jD1mrPxU%2Br5MR1T6Ov5a0nITz%0D%0AMenbqv%2BxnD90mn%2BnZjfljG</a></td>
</tr>
<tr>
<td>Update profile</td>
<td>user=ken</td>
<td><a href="http://server/user/?op=update&amp;hashData=QW75YSxM0spQicCP%2Bljn%2Lbcy1vZjy2cv6%2B0vR8kHFRilGrLOmEw5L4EHf%2B2BvxVclIftinV%2BCUP%0D%0AhVXZnetX%2Fw%3D%3D">http://server/user/?op=update&amp;hashData=QW75YSxM0spQicCP%2Bljn%2Lbcy1vZjy2cv6%2B0vR8kHFRilGrLOmEw5L4EHf%2B2BvxVclIftinV%2BCUP%0D%0AhVXZnetX%2Fw%3D%3D</a></td>
</tr>
</tbody>
</table>

Table 3.1: REST calls passed from the Dicode workbench to the Collaboration services

### 3.2 Suite of services integrated

In this section, the services already integrated within the Dicode workbench are presented. Currently, two different types of services have been successfully integrated: first, a set of services aimed to deal with the management issues of the Dicode workbench, such as users, services and workspaces management or login process. Second, a set of services oriented to support collaboration processes among different users with the same topics of interests.

All these services have been implemented as RESTful services (Fielding, 2000). These services are detailed in the following sub-sections.

#### 3.2.1 Workbench management services

As mentioned before, the Dicode workbench has been implemented as a web application using Java technologies and MySQL database management system. To manage some features within the workbench, different RESTful services have been designed. Those features have been divided into three main categories:

- **Users.** To manage all the information related with the Dicode workbench users. This ranges from registry of users or login processes to information updates.
- **Services.** This category includes those services aimed to manage the information of the services published by the service providers in the Dicode workbench and stored in the Dicode Registry of services described in section 3.1.2.
- **Workspaces.** These services manage the information related with the shared workspaces created by users within the Dicode workbench.
All these services are detailed in the next sub-sections, following the abstract service specification schema proposed in Appendix B of deliverable D3.1.1 - The Dicode Data Mining Framework (initial version).\(^5\)

Regarding the errors returned by these services, Table 3.2 shows the error codes shared by all of them:

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td><strong>Database connection error.</strong> This error occurs when the REST service tries to get connection with the database but for some reason is unable to do it.</td>
</tr>
<tr>
<td>-2</td>
<td><strong>SQL Exception.</strong> This error occurs when the REST service tries to execute a SQL sentence that is incomplete in some point.</td>
</tr>
<tr>
<td>-3</td>
<td><strong>Bad Query.</strong> This error occurs when the REST service tries to execute a SQL query that is made with errors or it is incomplete.</td>
</tr>
<tr>
<td>-4</td>
<td><strong>Integrity constraint violation.</strong> This error occurs when the REST service tries to make an update that endangers the integrity of the database.</td>
</tr>
<tr>
<td>-5</td>
<td><strong>I/O Exception.</strong> This error occurs when the REST service tries to make an in/out operation but this operation, for some reason, cannot be performed.</td>
</tr>
<tr>
<td>-6</td>
<td><strong>Encoding Error.</strong> This error occurs when the REST service tries to encode characters, and for some reason, the encoding goes wrong.</td>
</tr>
</tbody>
</table>

Table 3.2: List of error codes of the Dicode workbench management services

3.2.1.1 User management service

In this section, a coarse description of the Dicode services dealing with information about Dicode workbench users is provided.

*Note:* this specification represents a snapshot of the current considerations. Forthcoming evolution of standards or changes in the system architecture may cause future modifications.

<table>
<thead>
<tr>
<th>Name</th>
<th>User management service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standards</td>
<td>Related standards are:</td>
</tr>
<tr>
<td></td>
<td>• REST (Fielding, 2000)</td>
</tr>
<tr>
<td></td>
<td>• XML (W3C – Extensible Markup Language, 2006)</td>
</tr>
<tr>
<td>Description</td>
<td>This service provides all functionalities needed to manage users within the Dicode workbench. Those functionalities include access to databases containing the data.</td>
</tr>
<tr>
<td></td>
<td>The user management service defines the following interfaces:</td>
</tr>
<tr>
<td></td>
<td>• <em>Users:</em> The users interface manages all the referent with the users in the database.</td>
</tr>
</tbody>
</table>

The way to access the service is as follows:
http://${server}:${port}/ {Interface}/{Operation}/{parameters}

<table>
<thead>
<tr>
<th>Interface</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>users</td>
<td>validateUser</td>
</tr>
</tbody>
</table>

This operation validates users into the system. It checks user’s username and user’s password.

**Input:**

- **userName**: String. Correspond with the user’s username.
- **password**: String. Correspond with the user’s password.

**Output:**

- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
http://${server}:${port}/users/validateUser/{userName}/{password}

<table>
<thead>
<tr>
<th>Interface</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>users</td>
<td>isUser</td>
</tr>
</tbody>
</table>

This operation checks if one user is already registered in the system.

**Input:**

- **userName**: String. Corresponds with the user’s username.

**Output:**

- 0 if an error occurred.
- **Number greater than 0** if Ok. This number corresponds with the user’s ID.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
http://${server}:${port}/users/isUser/{userName}

<table>
<thead>
<tr>
<th>Interface</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>users</td>
<td>isMail</td>
</tr>
</tbody>
</table>

This operation checks if one email is already registered in the system.

**Input:**

- **mail**: String. Corresponds with the user’s email.

**Output:**

- 0 if an error occurred.
- **Number greater than 0** if Ok. This number corresponds with the user’s ID.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
http://${server}:${port}/users/isMail/{mail}

<table>
<thead>
<tr>
<th>Interface</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>users</td>
<td>addUser</td>
</tr>
</tbody>
</table>

This operation adds one user into the data base.
### Input:
- **fullName**: String. Corresponds with the user’s full name.
- **userName**: String. Corresponds with the user’s user name.
- **password**: String. Corresponds with the user’s password.
- **organization**: String. Corresponds with the user’s organization.
- **mail**: String. Corresponds with the user’s email.
- **status**: Int. Corresponds with the user’s status.

### Output:
- 0 if an error occurred.
- **Number greater than 0** if Ok. This number corresponds with the user’s ID.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/users/addUser/{fullName}/{userName}/{password}/{organization}/{mail}/{status}
```

### activateUser
This operation activates one user in the system.

**Input:**
- **userID**: Int. Corresponds with the user’s ID.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/users/activateUser/{userID}
```

### updateUser
This operation updates the information for one user.

**Input:**
- **userID**: Int. Corresponds with the user’s ID.
- **fullName**: String. Corresponds with the user’s full name.
- **userName**: String. Corresponds with the user’s user name.
- **organization**: String. Corresponds with the user’s organization [optional]
- **mail**: String: Corresponds with the user’s email.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/users/updateUser/{userID}/{fullName}/{userName}/
```
### updateState

This operation updates the information of the one user’s state.

**Input:**
- `userID`: Int. Corresponds with the user’s ID.
- `status`: Int. Corresponds with the user’s status.

**Output:**
- `1` if OK.
- `0` if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/users/updateState/{userID}/{status}
```
### D5.4.1 (version 5):

#### Input:
- This operation doesn’t receive any parameter.

#### Output:
This operation returns a XML list with the next relevant tags for each rol:
- `<id>`: it contains the rol’s id.
- `<name>`: it contains the rol’s name.

This operation returns a XML list with the next relevant tags for each permission:
- `<id>`: it contains the permission identification.
- `<name>`: it contains the name of the permission.
- `<description>`: it contains the description of the permission.

The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/users/listRoles
```

<table>
<thead>
<tr>
<th>rolesUser</th>
<th>This operation shows all the one user’s roles.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td></td>
</tr>
<tr>
<td>• userID:</td>
<td>Int. Corresponds with the user’s ID.</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td></td>
</tr>
<tr>
<td>This operation returns a XML list with the next relevant tags for each rol:</td>
<td></td>
</tr>
<tr>
<td>• <code>&lt;id&gt;</code>: it contains the identification of the rol.</td>
<td></td>
</tr>
<tr>
<td>The output of the operation is in the format “text/xml”.</td>
<td></td>
</tr>
<tr>
<td>The way to invoke this operation is as follows:</td>
<td><code>http://${server}:${port}/users/rolesUser/{userID}</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>listStates</th>
<th>This operation shows the relevant information for all the states in the system.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td></td>
</tr>
<tr>
<td>• This operation doesn’t receive any parameter.</td>
<td></td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td></td>
</tr>
<tr>
<td>This operation returns a XML list with the next relevant tags for each state:</td>
<td></td>
</tr>
<tr>
<td>• <code>&lt;id&gt;</code>: it contains the identification of the state.</td>
<td></td>
</tr>
<tr>
<td>• <code>&lt;name&gt;</code>: it contains the name of the state.</td>
<td></td>
</tr>
<tr>
<td>• <code>&lt;description&gt;</code>: it contains the description of the state.</td>
<td></td>
</tr>
<tr>
<td>The output of the operation is in the format “text/xml”.</td>
<td></td>
</tr>
<tr>
<td>The way to invoke this operation is as follows:</td>
<td><code>http://${server}:${port}/users/listStates</code></td>
</tr>
<tr>
<td>Operation</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>listPrivileges</td>
<td>This operation shows the relevant information for all the permissions in the system.</td>
</tr>
<tr>
<td>Input:</td>
<td>• This operation doesn’t receive any parameter.</td>
</tr>
<tr>
<td>Output:</td>
<td>This operation returns a XML list with the next relevant tags for each permission:</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;id&gt;</code>: it contains the identification of the permission.</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;name&gt;</code>: it contains the name of the permission.</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;description&gt;</code>: it contains the description of the permission.</td>
</tr>
<tr>
<td></td>
<td>The output of the operation if in the format “text/xml”</td>
</tr>
<tr>
<td></td>
<td>The way to invocate this operation is as follows:</td>
</tr>
<tr>
<td></td>
<td>http://${server}:${port}/users/listPrivileges</td>
</tr>
<tr>
<td>deleteAllRoles</td>
<td>This operation deletes all the roles of one user.</td>
</tr>
<tr>
<td>Input:</td>
<td>• userID: Int. Corresponds with the user’s ID.</td>
</tr>
<tr>
<td>Output:</td>
<td>• 1 if OK.</td>
</tr>
<tr>
<td></td>
<td>• 0 if an error occurred.</td>
</tr>
<tr>
<td></td>
<td>The output of the operation is in the format “text/plain”</td>
</tr>
<tr>
<td></td>
<td>The way to invocate this operation is as follows:</td>
</tr>
<tr>
<td></td>
<td>http://${server}:${port}/users/deleteAllRoles/{userID}</td>
</tr>
<tr>
<td>deleteRole</td>
<td>This operation deletes one role of one user.</td>
</tr>
<tr>
<td>Input:</td>
<td>• userID: Int. Corresponds with the user’s ID.</td>
</tr>
<tr>
<td></td>
<td>• rolID: Int. Corresponds with the user’s ID.</td>
</tr>
<tr>
<td>Output:</td>
<td>• 1 if OK.</td>
</tr>
<tr>
<td></td>
<td>• 0 if an error occurred.</td>
</tr>
<tr>
<td></td>
<td>The output of the operation is in the format “text/plain”.</td>
</tr>
<tr>
<td></td>
<td>The way to invocate this operation is as follows:</td>
</tr>
<tr>
<td></td>
<td>http://${server}:${port}/users/deleteRole/{userID}/{rolID}</td>
</tr>
</tbody>
</table>
### deleteUser

This operation deletes one user from the system.

**Input:**
- **userID:** Int. Corresponds with the user’s ID.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/users/deleteUser/{userID}
```

### updatePass

This operation updates the password of one user.

**Input:**
- **userID:** Int. Corresponds with the user’s ID.
- **password:** String. Corresponds with the user’s password.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/users/updatePass/{userID}/{password}
```

### userState

This operation shows the information of the user’s state.

**Input:**
- **userID:** Int. Corresponds with the user’s ID.

**Output:**
This operation returns a XML code with the next relevant tags for the user’s state:
- `<id>`: It contains the userstate identification.
- `<name>`: It contains the name of the user state.
- `<description>`: It contains the userstate description.

The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/users/userState/{userID}
```
<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Input</th>
<th>Output</th>
<th>Invocation URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>addConfirmation</td>
<td>This operation adds a new confirmation in the data base.</td>
<td>• userID: Int. Corresponds with the user’s ID.</td>
<td>• 1 if OK.</td>
<td>http://${server}:${port}/users/addConfirmation/{userID}/{key}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• key: Int. Corresponds with the confirmation key.</td>
<td>• 0 if an error occurred.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The output of the operation is in the format “text/plain”.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The way to invoke this operation is as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>http://${server}:${port}/users/addConfirmation/{userID}/{key}</td>
<td></td>
</tr>
<tr>
<td>deleteConfirmation</td>
<td>This operation deletes one confirmation for one user from the data base.</td>
<td>• userID: Int. Corresponds with the user’s ID.</td>
<td>• 1 if OK.</td>
<td>http://${server}:${port}/users/deleteConfirmation/{userID}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0 if an error occurred.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The output of the operation is in the format “text/plain”.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The way to invoke this operation is as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>http://${server}:${port}/users/deleteConfirmation/{userID}</td>
<td></td>
</tr>
<tr>
<td>validateConfirmation</td>
<td>This operation validates the confirmation of one user.</td>
<td>• userID: Int. corresponds with the user’s ID.</td>
<td>• 1 if OK.</td>
<td>http://${server}:${port}/users/validateConfirmation/{userID}/{key}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• key: Int. Corresponds with the confirmation key.</td>
<td>• 0 if an error occurred.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The output of the operation is in the format “text/plain”.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The way to invoke this operation is as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>http://${server}:${port}/users/validateConfirmation/{userID}/{key}</td>
<td></td>
</tr>
</tbody>
</table>
### consultUsers

This operation shows all the relevant information for all the users registered in the system.

**Input:**
- This operation doesn’t receive any parameters.

**Output:**
This operation returns a XML list with the next relevant information tags for each user:
- `<id>`: It contains the user’s ID.
- `<fullName>`: It contains the user’s full name.
- `<userName>`: It contains the user’s username.
- `<organization>`: It contains the user’s organization.
- `<email>`: It contains the user’s email.
- `<stateID>`: It contains the user’s state identification.

The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/users/consultUsers
```

### newActivationDate

This operation changes the user’s activation date for the user given.

**Input:**
- `userID`: Int. Corresponds with the user’s ID.

**Output:**
- `1` if OK.
- `0` if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/users/newActivationDate/{userID}
```

### Example usage

This service will be used by all users every time they log in into the system. The user fills the login information (username and password) which it is sent to the login access process. This login process uses these services to check whether the data are correct and the user is granted to access the system.

### Comments

The Rest Services for Dicode Data Base access have been implemented using the open source framework Jersey. Jersey is the production quality, JAX-RS (JSR 311) Reference Implementation for building RESTful Web services.

Status of implementation: **implemented**

### Conformance classes

N/A

### Implementation rules

N/A
### 3.2.1.2 Service management service

In this section, a coarse description of the Dicode services dealing with information about services published within the Dicode workbench is provided.

*Note:* this specification represents a snapshot of the current considerations. Forthcoming evolution of standards or changes in the system architecture may cause future modifications.

<table>
<thead>
<tr>
<th>Name</th>
<th>Service management service</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standards</strong></td>
<td>Related standards are:</td>
</tr>
<tr>
<td></td>
<td>• REST (Fielding, 2000)</td>
</tr>
<tr>
<td></td>
<td>• XML (W3C – Extensible Markup Language, 2006)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This service provides all functionalities needed to manage services within the Dicode workbench. Those functionalities include access to databases containing the data.</td>
</tr>
<tr>
<td></td>
<td>The service management service defines the following interfaces:</td>
</tr>
<tr>
<td></td>
<td>• <em>Services</em>: The users interface manages all the referent with the users in the database.</td>
</tr>
<tr>
<td></td>
<td>The way to access the service is as follows: http://${server}:${port}/ {Interface}/{Operation}/{parameters}</td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td>services</td>
</tr>
<tr>
<td><strong>listServices</strong></td>
<td>This operation lists all the services IDs and names registered in the system.</td>
</tr>
<tr>
<td></td>
<td><strong>Input:</strong></td>
</tr>
<tr>
<td></td>
<td>• This operation doesn’t receive any parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>Output:</strong></td>
</tr>
<tr>
<td></td>
<td>This operation returns a XML list with the next relevant information tags for all the services in the system:</td>
</tr>
<tr>
<td></td>
<td>• <em>&lt;id&gt;</em>: It contains the service ID.</td>
</tr>
<tr>
<td></td>
<td>• <em>&lt;name&gt;</em>: It contains the service name.</td>
</tr>
<tr>
<td></td>
<td>The output of the operation is in the format “text/xml”.</td>
</tr>
<tr>
<td></td>
<td>The way to invocate this operation is as follows: http://${server}:${port}/services/listServices</td>
</tr>
<tr>
<td><strong>service</strong></td>
<td>This operation shows all the information for one service in the data base.</td>
</tr>
<tr>
<td></td>
<td><strong>Input:</strong></td>
</tr>
</tbody>
</table>
• **serviceID**: Int. Corresponds with the service ID.

**Output:**
This operation return a XML code with the next relevant information tags:
- `<id>`: It contains the service ID.
- `<uploaderUserID>`: It contains the uploader user ID.
- `<name>`: It contains the service name.
- `<description>`: It contains the service description.
- `<URI>`: It contains the service URI.
- `<creationDate>`: It contains the creation date of the service.
- `<modificationDate>`: It contains the last modification date.
- `<isActive>`: It contains the service active value.
- `<stateID>`: It contains the state of the service.
- `<invocationMethodID>`: It contains the service invocation method identification.
- `<typeID>`: It contains the service type ID.
- `<categoryID>`: It contains the service category ID.

The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:
http://${server}:${port}/services/service/{serviceID}

**isService**
This operation checks if a service is already registered in the system.

**Input:**
- **Name**: String. It corresponds with the service name.
- **URI**: String. It corresponds with the service URI.

**Output:**
- Service ID if OK.
- 0 if an error occurred.

The output of the operation is in format “text/plain”.

The way to invoke this operation is as follows:
http://${server}:${port}/services/isService/{Name}?URI={serviceUri}

**listDataTypes**
This operation lists all the data types registered in the system.

**Input:**
- This operation does not receive any parameter.

**Output:**
This operation returns a XML list with the next relevant information tags:
- `<id>`: It contains the data type ID.
- `<name>`: It contains the data type name.
- `<description>`: It contains the data type description.
- `<format>`: It contains the data type format.
The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:

http://${server}:${port}/services/listDataTypes

**listParamTypes**

This operation lists all the possible parameter types in the system for one type of service in particular.

**Input:**
- serviceTypeID: Int. It corresponds with the service type ID.

**Output:**
This operation returns a XML list with the following relevant information tags:
- <id>: It contains the parameter type ID.
- <name>: It contains the parameter type name.
- <description>: It contains the parameter type description.

The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:

http://${server}:${port}/services/listParamTypes/{serviceTypeID}

**listInvocationMethods**

This operation lists all the invocation methods registered in the system for one type of service in particular.

**Input:**
- serviceTypeID: Int. It corresponds with the service type ID.

**Output:**
This operation returns a XML list with the following relevant information tags:
- <id>: It contains the invocation method ID.
- <name>: It contains the invocation method name.
- <description>: It contains the invocation method description.

The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:

http://${server}:${port}/services/listInvocationMethods/{serviceTypeID}

**paramType**

This operation returns the parameter type for one parameter given.

**Input:**
- paramID: Int. It corresponds with the parameter ID.

**Output:**
This operation returns a XML code with the following relevant information tags:
- <name>: It contains the name of the parameter type.
- <description>: It contains the description of the parameter type.
The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:
http://${server}:${port}/services/paramType/{paramID}

<table>
<thead>
<tr>
<th><strong>dataType</strong></th>
<th>This operation returns the data type information for one parameter given.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>• paramaID: Int. It corresponds with the parameter ID.</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td>This operation returns a XML code with the following relevant information tags:</td>
</tr>
<tr>
<td></td>
<td>• &lt;name&gt;: It contains the data type name.</td>
</tr>
<tr>
<td></td>
<td>• &lt;description&gt;: It contains the data type description.</td>
</tr>
<tr>
<td></td>
<td>• &lt;format&gt;: It contains the data type format.</td>
</tr>
</tbody>
</table>

The output of the service is in the format “text/xml”.

The way to invoke this operation is as follows:
http://${server}:${port}/services/dataType/{paramID}

<table>
<thead>
<tr>
<th><strong>orderParams</strong></th>
<th>This operation orders the input parameters for one service given.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>• serviceID: It corresponds with the service ID.</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td>This operation returns a XML list with the following relevant information tags:</td>
</tr>
<tr>
<td></td>
<td>• &lt;id&gt;: It contains the parameter ID.</td>
</tr>
<tr>
<td></td>
<td>• &lt;name&gt;: It contains the parameter name.</td>
</tr>
<tr>
<td></td>
<td>• &lt;position&gt;: It contains the position of the parameter in the service call.</td>
</tr>
<tr>
<td></td>
<td>• &lt;paramTypeID&gt;: It contains de parameter type ID.</td>
</tr>
</tbody>
</table>

The output of the service is in the format “text/xml”.

The way to invoke this operation is as follows:
http://${server}:${port}/services/orderParams/{serviceID}

<table>
<thead>
<tr>
<th><strong>invocationMethod</strong></th>
<th>This operation shows all the invocation method information for one service given.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>• serviceID: It corresponds with the service ID.</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td>This operation returns a XML code with the following relevant information tags:</td>
</tr>
</tbody>
</table>
- `<name>`: It contains the invocation method name.
- `<description>`: It contains the invocation method description.

The output of the service is in the format “text/xml”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/services/invocationMethod/{serviceID}
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addService</td>
<td>This operation adds a new service into the data base.</td>
</tr>
<tr>
<td>Input:</td>
<td></td>
</tr>
<tr>
<td>uploaderUser:</td>
<td>Int. It corresponds with the ID of the user that uploads the service.</td>
</tr>
<tr>
<td>name:</td>
<td>String. It corresponds with the service name.</td>
</tr>
<tr>
<td>description:</td>
<td>String. It corresponds with the service description.</td>
</tr>
<tr>
<td>URI:</td>
<td>String. It corresponds with the service invocation URI.</td>
</tr>
<tr>
<td>tags:</td>
<td>String. It corresponds with the service tags.</td>
</tr>
<tr>
<td>active:</td>
<td>Int. It corresponds with the service active state.</td>
</tr>
<tr>
<td>serviceState:</td>
<td>Int. It corresponds with the service state.</td>
</tr>
<tr>
<td>invocationID:</td>
<td>Int. It corresponds with the invocation method ID.</td>
</tr>
<tr>
<td>typeID:</td>
<td>Int. It corresponds with the service type ID.</td>
</tr>
<tr>
<td>categoryID:</td>
<td>Int. It corresponds with the service category ID.</td>
</tr>
<tr>
<td>Output:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Service ID if OK.</td>
</tr>
<tr>
<td></td>
<td>0 if an error occurred.</td>
</tr>
</tbody>
</table>

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/services/addService/{uploaderUser}/{name}/
{active}/{serviceState}/{invocationID}/{typeID}/{categoryID}?description={description}&tags={tags}&URI={URI}
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateActive</td>
<td>This operation updates the active state of one service given.</td>
</tr>
<tr>
<td>Input:</td>
<td></td>
</tr>
<tr>
<td>serviceID:</td>
<td>Int. It corresponds with the service ID.</td>
</tr>
<tr>
<td>value:</td>
<td>Int. It corresponds with the new value for the active state.</td>
</tr>
<tr>
<td>Output:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 if OK.</td>
</tr>
<tr>
<td></td>
<td>0 if an error occurred.</td>
</tr>
</tbody>
</table>

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/services/updateActive/{serviceID}/{value}
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addParameter</td>
<td>This operation adds a new parameter in the data base.</td>
</tr>
</tbody>
</table>
### addParameter

**Input:**
- **name**: String. It corresponds with the name of the parameter.
- **position**: Int. It corresponds with the position of the parameter.
- **dataTypeID**: Int. It corresponds with the parameter data type ID.
- **parameterTypeID**: Int. It corresponds with the parameter type ID.
- **serviceID**: Int. It corresponds with the parameter service ID.

**Output:**
- 1 if OK
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/services/addParameter/{name}/{position}/{dataTypeID}/{parameterTypeID}/{serviceID}
```

### deleteParameter

This operation deletes one parameter of the data base.

**Input:**
- **serviceID**: Int. It corresponds with the parameter service ID.
- **parameterName**: String. It corresponds with the parameter name.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/services/deleteParameter/{serviceID}/{parameterName}
```

### listServiceTypes

This operation lists all the service types in the data base.

**Input:**
- This operation does not receive any parameter.

**Output:**
This operation returns a XML list with the following relevant information tags:
- `<id>`: It contains the service type ID.
- `<name>`: It contains the service type name.
- `<description>`: It contains the service type description.

The output of the operation is in the format “text/xml”.

---

D5.4.1 (version 5): Page 31 of 85
### updateURI

This operation updates the URI for one service given.

**Input:**
- **serviceID:** Int. It corresponds with the service ID.
- **URI:** String. It corresponds with the service URI.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/services/updateURI/{serviceID}?URI={URI}
```

### addServiceReturnsDataType

This operation adds a new return type for one service given.

**Input:**
- **name:** String. It corresponds with the name of the return type.
- **serviceID:** Int. It corresponds with the service ID.
- **dataTypeID:** Int. It corresponds with the data type ID.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/services/addServiceReturnsDataType/{name} /[serviceID]/{dataTypeID}
```

### updateInvocationMethod

This operation updates the invocation method for one service given.

**Input:**
- **serviceID:** Int. It corresponds with the service ID.
- **invocationMethodID:** Int. It corresponds with the invocation method ID.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/services/updateInvocationMethod/{serviceID} /{invocationMethodID}
```
### updateDataType

This operation updates the information for one data type.

**Input:**
- **dataTypeID**: Int. It corresponds with the data type ID.
- **name**: String. It corresponds with the new name for the data type.
- **description**: String. It corresponds with the new description for the data type.

**Output:**
- 1 if OK
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/services/updateDataType/{dataTypeID}?name={name}
```

### listReturnTypes

This operation lists all the return types in the database for one service given.

**Input:**
- **serviceID**: Int. It corresponds with the service ID.

**Output:**
This operation returns a XML list with the following relevant information tags:
- `<name>`: It contains the return type name.
- `<dataTypeID>`: It contains the data type ID for the return type.

The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/services/listReturnTypes/{serviceId}
```

### dataTypeData

This operation shows all the information for one data type given.

**Input:**
- **dataTypeID**: Int. It corresponds with the data type ID.

**Output:**
This operation return a XML code with the following relevant information tags:
- `<name>`: It contains the name of the data type.
- `<description>`: It contains the data type description.
- `<format>`: It contains the data type format.

The output of the service is in the format “text/xml”.

The way to invoke this operation is as follows:
searchService

This operation searches services according to the search parameters.

**Input:**
- **params**: List<String>. It’s a list with the search parameters.

**Output:**
This operation returns a XML list with the following relevant information tags:
- `<id>`: It contains the service ID.
- `<name>`: It contains the service name.

The output of this operation is in the format “text/xml”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/services/searchService?params={param1}&params={param2}&params={param3}&params={param4}&params={param5}
```

**Example usage**
This service is used by the Dicode workbench to present the information about services to the users. When users click on the “services” option of the menu box, the workbench invokes these services to retrieve the information about services registered in the system. This information is formatted and presented to the user.

**Comments**
The Rest Services for Dicode Data Base access have been implemented using the open source framework Jersey. Jersey is the production quality, JAX-RS (JSR 311) Reference Implementation for building RESTful Web services.

Status of implementation: **implemented**

**Conformance classes**
N/A

**Implementation rules**
N/A

**UML model**
N/A

### 3.2.1.3 Workspace management service

In this section, a coarse description of the Dicode services managing information about workspaces within the Dicode workbench is provided.

*Note*: this specification represents a snapshot of the current considerations. Forthcoming evolution of standards or changes in the system architecture may cause future modifications.

<table>
<thead>
<tr>
<th>Name</th>
<th>Workspace management service</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standards</strong></td>
<td>Related standards are:</td>
</tr>
<tr>
<td></td>
<td>• REST (Fielding, 2000)</td>
</tr>
</tbody>
</table>
**Description**

This service provides all functionalities needed to manage workspaces within the Dicode workbench. Those functionalities include access to databases containing the data.

The workspace management service defines the following interfaces:

- **Workspaces**: The users interface manages all the referent with the users in the database.

The way to access the service is as follows:

```
http://${server}:${port}/ {Interface}/{Operation}/{parameters}
```

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Workspaces</strong></td>
<td>This operation lists all the entry types in the system.</td>
</tr>
<tr>
<td><strong>listEntryTypes</strong></td>
<td>This operation lists all the entry types in the system.</td>
</tr>
<tr>
<td><strong>Input</strong>:</td>
<td>This operation does not receive any parameter.</td>
</tr>
<tr>
<td><strong>Output</strong>:</td>
<td>This operation returns a XML list with the following relevant information tags:</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;name&gt;</code>: It contains the entry type name.</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;description&gt;</code>: It contains the entry type description.</td>
</tr>
<tr>
<td></td>
<td>The output of the operation is in the format “text/xml&quot;.</td>
</tr>
<tr>
<td></td>
<td>The way to invocate this operation is as follows:</td>
</tr>
<tr>
<td></td>
<td><code>http://${server}:${port}/workspaces/listEntryTypes</code></td>
</tr>
<tr>
<td><strong>orderEntries</strong></td>
<td>This operation orders all the log entries for one workspace given.</td>
</tr>
<tr>
<td><strong>Input</strong>:</td>
<td>workspaceID: Int. It corresponds with the workspace ID.</td>
</tr>
<tr>
<td><strong>Output</strong>:</td>
<td>This operation return a XML list with the following relevant information tags:</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;authorID&gt;</code>: It contains the user ID for the user who wrote the log entry.</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;timeStamp&gt;</code>: It contains the log entry creation date.</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;message&gt;</code>: It contains the log entry message.</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;answers&gt;</code>: It contains the number of answers for the log entry.</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;typeID&gt;</code>: It contains the log entry type ID.</td>
</tr>
<tr>
<td></td>
<td>The output for this operation is in the format “text/xml&quot;.</td>
</tr>
<tr>
<td></td>
<td>The way to invocate this operation is as follows:</td>
</tr>
<tr>
<td></td>
<td><code>http://${server}:${port}/workspaces/orderEntries/{workspaceID}</code></td>
</tr>
<tr>
<td>Order Entries In Type</td>
<td>This operation orders all the log entries in one entry type for one workspace.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Input:                | • workspaceID: Int. It corresponds with the workspace ID.  
                         • entryTypeID: Int. It corresponds with the entry type ID. |
| Output:               | This operation return a XML list with the following relevant information tags: |
|                       | • <authorID>: It contains the user ID for the user who wrote the log entry. |
|                       | • <timeStamp>: It contains the log entry creation date.  
                         • <message>: It contains the log entry message.  
                         • <answers>: It contains the number of answers for the log entry.  
                         • <typeID>: It contains the log entry type ID. |
|                       | The output for this operation is in the format “text/xml”. |
|                       | An example of invocation could be as follows: http://${server}:${port}/workspaces/orderEntriesInType/{workspaceID}/{entryTypeID} |

<table>
<thead>
<tr>
<th>Add Workspace</th>
<th>This operation adds a new workspace into the data base.</th>
</tr>
</thead>
</table>
| Input:                | • name: String. It corresponds with the workspace name.  
                         • description: String. It corresponds with the workspace description.  
                         • tags: String. It corresponds with the workspace tags.  
                         • creatorUser: Int. It corresponds with the creator user of the workspace.  
                         • visibilityID: Int. It corresponds with the workspace visibility.  
                         • logBookID: Int. It corresponds with the workspace log book ID.  
                         • domain: Int. It corresponds with the workspace domain. |
| Output:               | • 1 if OK.  
                         • 0 if an error occurred. |
|                       | The output of this operation is in the format “text/plain”. |
|                       | The way to invoke this operation is as follows:  
                         http://${server}:${port}/workspaces/addWorkspace/{name}/{description}/{creatorUser}/{visibility}/{logBook}/{domain}?tags={tags} |

<table>
<thead>
<tr>
<th>Order Answers</th>
<th>This operation orders the answers for one log entry.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>• entryID: Int. It corresponds with the log entry ID.</td>
</tr>
</tbody>
</table>
### Output:
This service returns a XML list with the following relevant information tags:

- `<authorID>`: It contains the user ID for the author of the answer.
- `<timestamp>`: It contains the answer creation date.
- `<message>`: It contains the answer message.
- `<answers>`: It contains the answers number for the answer.
- `<typeID>`: It contains the answer log entry type ID.

The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/workspaces/orderAnswers/{entryID}
```

### addLogEntry
This operation adds a new log entry in the data base.

### Input:

- `message`: String. It corresponds with the log entry message.
- `Answers`: Int. It corresponds with the log entry number of answers.
- `logBookID`: Int. It corresponds with the logbook ID.
- `authorID`: Int. It corresponds with the ID of the user who creates the log entry.
- `parentEntryID`: Int. It corresponds with the parent log entry ID.
- `typeID`: Int. It corresponds with the log entry type ID.

### Output:

- 1 if OK.
- 0 if an error occurred.

The output of this operation is in the format “text/plain”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/workspaces/addLogEntry/{message}/{answers}/{logBookID}/{authorID}/{parentEntryID}/{typeID}
```

### workspace
This operation shows all the information contained in the data base for one workspace.

### Input:

- `workspaceID`: Int. It corresponds with the workspace ID.

### Output:
This operation returns a XML code with the following relevant information tags:

- `<name>`: It contains the workspace name.
- `<description>`: It contains the workspace description.
- `<visibilityID>`: It contains the workspace visibility ID.
- `<creatorID>`: It contains the user ID of the user who creates the workspace.
<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>addLogBook</td>
<td>This operation adds a new log book in the database.</td>
<td>•  This operation does not receive any parameters.</td>
<td>•  1 if OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>•  0 if an error occurred.</td>
</tr>
<tr>
<td>listWorkspaces</td>
<td>This operation lists all the workspaces in the database.</td>
<td>•  This operation does not receive any parameter.</td>
<td>This operation returns an XML list with the following relevant information tags:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>•  <code>&lt;id&gt;</code>: It contains the workspace ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>•  <code>&lt;visibilityID&gt;</code>: It contains the workspace visibility ID.</td>
</tr>
<tr>
<td>listVisibilities</td>
<td>This operation lists all the workspace visibilities in the database.</td>
<td>•  This operation does not receive any parameter.</td>
<td>This operation returns an XML list with the following relevant information tags:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>•  <code>&lt;id&gt;</code>: It contains the visibility ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>•  <code>&lt;name&gt;</code>: It contains the visibility name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>•  <code>&lt;description&gt;</code>: It contains the visibility description.</td>
</tr>
</tbody>
</table>

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
http://${server}:${port}/workspaces/addLogBook
<table>
<thead>
<tr>
<th><strong>userPermissionInWorkspace</strong></th>
<th>This operation shows what users have permissions and what permissions they have about the workspace.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>• workspaceID: Int. It corresponds with the workspace ID.</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td>This operation return a XML list with the following relevant information tags:</td>
</tr>
<tr>
<td></td>
<td>• &lt;userID&gt;: It contains the user ID of one user who has permissions in the workspace.</td>
</tr>
<tr>
<td></td>
<td>• &lt;permissionID&gt;: It contains the permission ID for the user in the workspace.</td>
</tr>
<tr>
<td></td>
<td>• &lt;permissionValue&gt;: It contains the permission value for the user in the workspace.</td>
</tr>
<tr>
<td></td>
<td>• &lt;permissionDescription&gt;: It contains the permission description for the user in the workspace.</td>
</tr>
<tr>
<td></td>
<td>The output of this operation is in the format “text/xml”.</td>
</tr>
<tr>
<td></td>
<td>The way to invocate this operation is as follows:</td>
</tr>
<tr>
<td></td>
<td>http://${server}:${port}/workspaces/userPermissionInWorkspace/{workspaceID}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>listWorkspacePermissions</strong></th>
<th>This operation lists all the information in the data base for the workspace permissions.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>• This operation Does not receive any parameter.</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td>This operation returns a XML list with the following relevant information tags:</td>
</tr>
<tr>
<td></td>
<td>• &lt;id&gt;: It contains the workspace permission ID.</td>
</tr>
<tr>
<td></td>
<td>• &lt;value&gt;: It contains the workspace permission value.</td>
</tr>
<tr>
<td></td>
<td>• &lt;description&gt;: It contains the workspace permission description.</td>
</tr>
<tr>
<td></td>
<td>The output of this operation is in the format “text/xml”.</td>
</tr>
<tr>
<td></td>
<td>The way to invocate this operation is as follows:</td>
</tr>
<tr>
<td></td>
<td>http://${server}:${port}/workspaces/listWorkspacePermissions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>addUserPermissionInWorkspace</strong></th>
<th>This operation adds one workspace permission for one user into the data base.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>• userID: Int. It corresponds with the user ID.</td>
</tr>
<tr>
<td></td>
<td>• workspaceID: Int. It corresponds with the workspace ID.</td>
</tr>
<tr>
<td></td>
<td>• workspacePermissionID: Int. It corresponds with the</td>
</tr>
</tbody>
</table>
### Workspace Permission ID

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

An example of invocation could be as follows:
```
http://${server}:${port}/workspaces/addUserPermissionInWorkspace/{userID}/{workspaceID}/{workspacePermissionID}
```

### addWorkspace

This operation adds one workspace into the data base.

**Input:**
- **workspaceID**: Int. It corresponds with the workspace ID.
- **code**: Int. It corresponds with the workspace code.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/workspaces/addWorkspace/{workspaceID}/{code}
```

### modifyWorkspaceName

This operation changes the name of one workspace given.

**Input:**
- **workspaceID**: Int. It corresponds with the workspace ID.
- **newName**: String. It corresponds with the workspace new name.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
```
http://${server}:${port}/workspaces/modifyWorkspaceName/{workspaceID}/{newName}
```

### modifyWorkspaceDescription

This operation changes the description of one workspace given.

**Input:**
- **workspaceID**: Int. It corresponds with the workspace ID.
- **newDescription**: String. It corresponds with the workspace new description.
Output:
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “Text/plain”.

An example of invocation could be as follows:
http://${server}:${port}/workspaces/modifyWorkspaceDescription/{workspaceID}/{newDescription}

servicesInWorkspace
This operation lists all the services belonging to a workspace given.

Input:
- workspaceID: Int. It corresponds with the workspace ID.

Output:
This operation returns a XML list with the following relevant information tags:
- <serviceID>: It contains the service ID.
- <row>: It contains with the service row. That’s indicates the first service coordinate in the interface.
- <colID>: It contains with the service column id. That’s indicates the second service coordinate in the interface.

The output of the service is in the format “text/xml”.

The way to invoke this operation is as follows:
http://${server}:${port}/workspaces/servicesInWorkspace/{workspaceID}

colName
This operation returns the column name.

Input:
- colID: Int. It corresponds with the column ID.

Output:
This operation returns a XML code with the following relevant information tag:
- <name>: It contains the column name.

The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:
http://${server}:${port}/workspaces/colName/{colID}

changeServiceInWorkspace
This operation changes the coordinates of one service given in one workspace given.

Input:
- workspaceID: Int. It corresponds with the workspace ID.
- serviceID: Int. It corresponds with the service ID.
- row: Int. It corresponds with the service new row. The new first
### addServiceInWorkspace
This operation adds a new service into one workspace.

**Input:**
- **workspaceID:** Int. It corresponds with the workspace ID.
- **serviceID:** Int. It corresponds with the service ID.
- **row:** Int. It corresponds with the service row. The first coordinate.
- **colID:** Int. It corresponds with the service column. The second coordinate.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the service is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/workspaces/addServiceInWorkspace/{workspaceID}/{serviceID}/{row}/{colID}
```

### deleteServiceInWorkspace
This operation removes one service from one workspace given.

**Input:**
- **workspaceID:** Int. It corresponds with the workspace ID.
- **serviceID:** Int. It corresponds with the service ID.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/workspaces/deleteServiceInWorkspace/{workspaceID}/{serviceID}
```

### workspacesInWorkspace
This operation lists the workspaces’ identification for one workspace given.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>- workspaceID: It correspond with the workspace identificacion.</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td>This operation returns a XML list with the following relevant information tag:</td>
</tr>
<tr>
<td></td>
<td>- &lt;id&gt;: It contains the workspace identificacion.</td>
</tr>
<tr>
<td></td>
<td>The output of the operation is in the format “text/xml”.</td>
</tr>
<tr>
<td></td>
<td>The way to invoke this operation is as follows:</td>
</tr>
<tr>
<td></td>
<td>http://${server}:${port}/workspaces/workspacesInWorkspace/{workspaceID}</td>
</tr>
</tbody>
</table>

**workspaceInformation**

This operation lists the workspace information attributes for the workspace ID given.

**Input:**
- workspaceID: It correspond with the workspace Identificacion.

**Output:**
This operation returns a XML code with the following relevant information tags:
- <cView>: It contains the workspace cView.
- <fView>: It contains the workspace fView.
- <aView>: It contains the workspace aView.

The output of the operation is in the format “text/xml”.

The way to invoke this operation is as follows:
http://${server}:${port}/workspaces/workspaceInformation/{workspaceID}

**modifyRowCol**

This operation modifies the position of the service given in the workspace given.

**Input:**
- workspaceID: Int. It corresponds with the workspace ID.
- serviceID: Int. It corresponds with the service identificacion.
- newRow: Int. It corresponds with the new row position.
- newCol: Int. It corresponds with the new column position.

**Output:**
- 1 if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

An example of invocation could be as follows:
http://${server}:${port}/workspaces/modifyRowCol/{workspaceID}/
Example usage

This service is used by the Dicode workbench to present the information about workspaces to the users. When users click on the “workspace” option of the menu box, the workbench invokes these services to retrieve the information about workbenches registered in the system. This information is formatted and presented to the user.

Comments

The Rest Services for Dicode Data Base access have been implemented using the open source framework Jersey. Jersey is the production quality, JAX-RS (JSR 311) Reference Implementation for building RESTful Web services.

Status of implementation: implemented

Conformance classes

N/A

Implementation rules

N/A

UML model

N/A

3.2.2 Collaboration support services

3.2.2.1 Description

Within the Dicode project, collaboration services aim at providing innovative workspaces that allow users to facilitate sense- and decision making in data-intensive environments. As reported in deliverable D2.2: “The Dicode Approach User requirements, conceptual integrative architecture, agile methodology and functional specifications” in the context of the Dicode use cases, collaboration services provide the functionalities which will enable users to engage into argumentative collaboration in order to: i) select the appropriate data sources and processing techniques, ii) collaboratively interpret the outcome of the research facilitating sense-making and iii) support decision making towards selecting specific courses of action.

The initial version of collaboration services, whose technical specifications have been presented in deliverable D4.1.1: “The Dicode Collaboration Support Services (initial version)”, have been integrated into the Dicode workbench demonstrating how they can be used to support collaboration needs of the use cases. In particular, the initial implementation of the view based collaboration model as outlined in deliverable D2.2 has been integrated into the workbench, which include:

- **Discussion-forum view**, where collaboration workspaces are presented as traditional web-based forums, and user posts are displayed in ascending chronological order. Users are able to post new messages to the collaboration space, which appear at the end of the list of messages. Posts may also have attachments to enable the uploading of files. Collaboration monitoring services have been integrated into the forum view as well. In particular, the ability to find discussions, similar to the one displayed, is supported. The discussion-forum view of workspaces emphasizes and provides support for sharing of resources.

---

6 [http://dicode-project.eu/sites/default/files/D2.2-TheDicodeApproach-v6-EC_0.pdf](http://dicode-project.eu/sites/default/files/D2.2-TheDicodeApproach-v6-EC_0.pdf)

• **Mind-map view**, where the collaboration workspace is displayed as a mind map allowing users to interact with the available items. The mind-map deploys a spatial metaphor permitting the easy movement and arrangement of items on the collaboration space. Messages posted on the collaboration space in mind-map view are semantically typed and have predefined types (such as “idea”, “comment” or “note”). “Generic” types are also supported in case the semantics of a message is unclear. Files of any content type (e.g. pdf, jpg) can be uploaded to the collaboration space. The mind-map view provides also a set of mechanisms through which: (i) items on the collaboration space can be related, and (ii) new abstractions can be created. In particular, creation of relationships between items is possible by drawing directed arrows between items on the collaboration space. The visual characteristics of the arrows connecting items – such as their color and thickness - can be specified. In addition, labels can be associated to arrows indicating their meaning. Items on the collaboration space can also be aggregated, which allows a group of items to be treated as a single entity. Furthermore, aggregated/grouped items can be transformed into a single item of one of the available types, thereby creating new, composite items on the collaboration space. The mind-map view aims at supporting sense-making during cognitive complex tasks.

• **Formal/Argumentation view**, where the collaboration workspace accommodates reasoning algorithms that allow active support towards decision making. The formal view of the collaboration space permits only a limited set of discourse moves for a limited set of message types whose semantics is fixed. In particular, the formal view enables the posting of messages which can be of type “issue”, “alternative” or “position”. The formal view supports, in addition, the notion of preferences. A preference is used to weigh the importance of two positions and reflect the importance of one position over another. Specialized reasoning algorithms (such as algorithms based on voting, or algorithms based on multiple criteria), which are associated with the collaboration space, take into consideration the relationships of positions as well as existing preferences and calculate which alternative is currently prevailing or which position has been defeated. The aim of the formal view is to make the collaboration workspace machine understandable and further support decision making.

In the context of the Dicode workbench, the above mentioned collaboration services are available in all use case specific workbench workspaces, but are tailored to support the particular needs of the use case. More specifically, the set of available types in the mind-map view of collaboration workspaces differs per use case to support use case specific collaboration requirements. Likewise, the formal/argumentation view of workspaces can provide different reasoning algorithm, depending on each use case.

Collaboration services have been integrated into the Dicode workbench as widgets in which collaboration workspaces are displayed using the IFRAME HTML element (HTML IFRAME element, 2011).

### 3.2.2.2 Interface overview

In the next sections, we present the interfaces involved in integrating the collaboration services within the Dicode workbench. We present in particular the user interface, related to the way users experience and interact with the integrated collaboration services, and
technical interfaces, which aim to provide the necessary operations that the workbench can execute to facilitate a tighter integration with the collaboration services.

With respect to the user interface, as described previously, the collaboration services have been integrated into the Dicode workbench as widgets, which display collaboration workspaces using the IFRAME HTML element. The following images illustrate how the integration of collaboration services is visible to end users. A tab-based approach is used to visualize the previously described views of a single collaboration workspace: each tab is a widget displaying a different view of the same collaboration workspace. Figures 3.4, 3.5 and 3.6 show the different tabs displaying the mind-map, forum and formal view, respectively, of a collaboration workspace related to Use Case 1.

Figure 3.4: The “Mind-map view” tab of the Dicode workbench showing a collaboration workspace from Use Case 1 as a mind-map
**Figure 3.5:** The “Forum view” tab of the Dicode workbench shows the collaboration workspace of Figure 3.4 in forum view. The button “Relevant discussions” indicates integration of collaboration monitoring services.

**Figure 3.6:** The “Argumentation view” tab of the Dicode workbench shows the collaboration workspace of Figure 3.4 in formal/argumentation view.
Users can switch anytime between tabs and visualize the collaboration workspaces using the desired view. Each view supports the respective operations that permit users to participate in the collaboration. Moreover, the forum view of workspaces provides also integration with the collaboration monitoring services, which allow users to search for relevant discussions (Figure 3.5, button “Relevant discussions” at the top).

With respect to the technical interfaces, to further enable the tight and seamless integration between the Dicode workbench and the collaboration services, a number of additional REST-based services are currently under development. These REST-services augment the “Integration Interface” which was presented in deliverable D4.1.1: “The Dicode Collaboration Support Services (initial version)” and focus mainly on managing the available workspaces from within the Dicode workbench. These additional REST-based services are presented in the Table 3.3.

<table>
<thead>
<tr>
<th>REST service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateWorkspace</td>
<td>Allows the creation of a new workspace from within the Dicode workbench.</td>
</tr>
<tr>
<td>GetWorkspaces</td>
<td>Allows retrieval of workspaces associated with a use case. The call returns the Ids and title of the requested workspaces which can be displayed in the relevant workbench widget.</td>
</tr>
<tr>
<td>DeleteWorkspace</td>
<td>Permanently removes a workspace from the collaboration services.</td>
</tr>
</tbody>
</table>

Table 3.3: Additional REST-based services to integrate collaborative services within the Dicode workbench

### 3.2.3 Storage service

In this section, we describe the envisaged generic storage service designed to be used in the Dicode project. The objective of the Dicode project is to facilitate and augment collaboration and decision making in data-intensive and cognitively-complex settings. In such collaborative scenarios, users need to share and exchange information, files, reports, etc. The storage service arises as a solution for this necessity.

The main purpose of this service is to provide Dicode users with a permanent and reliable storage place to keep files accessible. The service will be as generic as possible to allow storing any kind of files (text plain, doc, pdf, html, xml, json, zip...). The service will provide mechanisms to upload files and retrieve them by using RESTful services. Additionally, meta-data information about files will be also stored to facilitate their search and location by search engines or services. These meta-data will contain information such as type of file (pdf, html, xml, etc.) or type of content (Dynamika report, DNA sequence, etc.). These types of files and contents should be enumerated in the Dicode ONtology (DON).

Within the Dicode consortium, the Technical Development Committee (TDC) discussed the way to proceed for the design and implementation of this service. Two different approaches were considered; centralized and distributed. After analyzing and discussing the benefits and drawbacks provided by each approach, we decided to develop the storage service as a

---

centralized repository. We considered that performance, reliability and availability of the service will be better than using a distributed approach.

Hence, at the moment, the storage service is envisaged as a centralized repository from the end-user perspective, but it will be designed following a mixed approach between centralized and distributed. When users want to use this service to share a file, they will have the option of uploading the file to the central repository or providing a URI accessible from the Internet where the file will be available. This mixed approach allows both to maintain privacy of files as to provide a storage place “in the cloud” to those users that cannot directly provide access to files from their organizations.

The scope for this storage service is thought for usage within the Dicode project, but the generic approach presented allows using it from outside, facilitating its reusability and scalability. At the moment of writing this deliverable, the service is being developed. A functional version will be integrated shortly.

3.2.3.1 Description
All services will be implemented by using RESTful services. The logical vision (end-user vision) of the storage service is illustrated in Figure 3.7.

![Figure 3.7: Logical vision of the storage service](image)

In this general scenario, one user needs to share a file and s/he decided to use the storage service. The file is uploaded and stored in a database in the cloud. The storage service assigns a unique identifier to the file. When another or the same user wants to retrieve the file, s/he invokes the storage service using such an identifier. This is what happens from the point of view of the end-user.

The storage service consists of three main components:

1. **Local database**, directly managed by the storage service. Used to physically store the files uploaded by the users. Local in this context means that it is running on the same server or cluster as the Dicode workbench.

2. **Metadata registry**, also directly managed by the storage service. Used to store the metadata about the files uploaded/published by the users. Such metadata contains information about the users, file annotations, creation and modification dates, etc.
3. **Semantic services** are used to annotate the files. The storage service only accesses this services that are deployed over the Internet to get the tags/classes of DON that are useful for annotation purposes.

As mentioned before, the storage service follows a mixed approach. When users want to share/upload a file, two different scenarios are possible:

- The user wants to store a file in a centralized repository directly managed by the storage service.
- The user wants to maintain the file and provides a public and accessible URI/reference to retrieve such file when required.

![Figure 3.8: Uploading a file to the centralized repository (scenario 1)](image)

The first scenario is presented in Figure 3.8. User wants to share a file but s/he cannot provide a permanent link to retrieve such file. So, s/he decides to use the storage service to upload the file. The necessary steps are the following: (1) the user contacts the storage service. (2) The storage service dynamically consults the tags related to file annotations. (3) This information is forwarded to the user that selects some of them and (4) sends them back to the storage service, together with the file itself and some extra meta-information. (5) The storage service creates a unique identifier for the new file and stores the file in its local database and (6) the metadata in the local registry. Finally, (7) the user is provided with the complete URI for the file.

The second scenario is depicted in Figure 3.9. In this case, the user does not want to upload the file anywhere because of, for instance, privacy concerns or legal issues. So, s/he decides to provide only the URI where the file is available. The process is the same as described for scenario 1 except step 5 because in this case, the file is not stored in the local database. The user sends the metadata to the storage service plus a reference/URI to the file. The storage...
service stores all this information in the metadata registry (including the reference to the file. Nothing is stored in the local database.

**Figure 3.9:** Uploading a reference to the file (scenario 2). Double green arrow represents that the file is accessible through the Internet. The rest of arrows have the same meaning as in Figure 3.8.

In both scenarios, metadata information about files is stored in the same registry. Particularly relevant are the metadata concerning the file format and content type. This information will be stored as tags using concepts from the DON. Hence, these concepts have to be previously defined in the DON.

To retrieve any file from the storage service, the user has only to indicate the file identifier assigned by the storage service. The storage service will implement the logic needed to retrieve the file wherever it is stored. This will be transparent for the end-user.

The storage service might be useful both to store user files and result, intermediate or temporal files coming from the execution of other services such as . In the last case, the users would be the application that uses the storage service. All kind of users will be able to:

- Upload and annotate files
- Download files
- Update already existing files and annotations
- Use stored files as input to data mining services
- List the existing files in a repository

All these functionalities are being implemented as RESTful services.

To ensure privacy of files, service storage will manage different repositories. For instance, in the Dicode workbench different workspaces (working areas) will be defined, one per use case. Each workspace might use the storage service but they will only have access to their own repository. Users of workspace 1 will not be able to see and access to the repository of workspace 2. The Dicode workbench will manage these privileges.
At the moment, the storage and access services are envisioned as services dealing with whole files. In the future the access service could be expanded to allow users to retrieve some concrete parts of the files.

3.2.3.2 Interface overview

In order to be integrated within the Dicode Workbench, a widget based interface will be developed. A preliminary design of such an interface is shown in Figure 3.10.

![Figure 3.10: Preliminary design of the widget-based interface of the storage service for the Dicode workbench](image)

This widget will display a generic label to identify the service on the top (“Storage service”). There will be a menu with, at least, three options:

- **“Upload file...”**, this option will allow users to upload files to the storage service. A new window (Figure 3.11) will be opened to capture the metadata information from users, and select the file and its location. Once the uploading process is finished successfully, the new file will be shown within the tree view.

- **“Configure”**, this option will allow users to configure some parameters of the storage service.

- **“About”**, this option will display information about the developers, dates, licenses and useful information about the service.

In the widget body, the list of available files will be shown using a tree view. Users will be able to retrieve the files by just clicking on the name. Additionally, the widget will allow dragging any file and dropping it over another service/widget within the workbench. Scrolls will be shown whenever needed.
As mentioned before, when users want to upload a file, they will click on the “Upload file...” option and a new window will appear. This window will contain a form to capture the metadata information about the file. A preliminary design of this form is presented in Figure 3.11.

![Uploading a file...](image)

**Figure 3.11:** Preliminary design of the form to capture the information about the file.

The form in Figure 3.11 presents the following fields to be completed by the user:

- **Name:** textual identifier of the file that will be used to display in the tree view.
- **Description:** textual description of the file and its contents.
- **File format:** this field allows users to specify the format of the file. This format will be selected from a fixed list. The list of supported file formats will be retrieved using the semantic services (DON).
- **File contents:** this field allows users to specify the contents of the file. These content types will be selected from a fixed list. The list of supported contents will be retrieved using the semantic services (DON). A mechanism/protocol should be defined to update and maintain the list of formats and contents supported in the DON.
- **File location:** the user can decide whether the file will be uploaded and stored in the cloud or will remain in its original location and a public URI will be provided to access it.
  - When user selects “In the cloud”, s/he has to select a file from his/her local machine by using the “Browse...” button.
  - If user selects “URI”, s/he must specify the public URI to the file.
Apart from this information, the Dicode workbench will send extra information to the storage service about the user uploading the file or the workspace/repository that the file belongs to.

### 3.2.3.3 Internal design

In this sub-section the preliminary internal design of the storage service is presented. Figure 3.12 depicts the Entity-Relationship (E/R) schema for the information managed by the service.

A coarse description of the Storage service interfaces and operations is provided. As previously, this information is presented following the abstract service specification schema proposed in Appendix B of D3.1.1: “The Dicode Data Mining Framework (initial version)”.

*Note:* this specification represents a snapshot of the current considerations. Forthcoming evolution of standards or changes in the system architecture may cause future modifications.

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Storage service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standards</td>
<td>Related standards are:</td>
</tr>
</tbody>
</table>

---

- REST (Fielding, 2000)
- XML (W3C – Extensible Markup Language, 2006)

### Description
This service provides all functionalities needed to allow storing files in the Dicode workbench.

The Storage service provides its functionality through the following interfaces:

- **DicodeStorage**: this interface implements the operations needed to store, locate and retrieve files from the repository.

The way to access the service is as follows:

```
http://${server}:${port}/{Interface}/{Operation}/{parameters}
```

### Interface

<table>
<thead>
<tr>
<th><strong>Interface</strong></th>
<th><strong>Operation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DicodeStorage</strong></td>
<td><strong>addFile</strong></td>
</tr>
</tbody>
</table>

This operation adds the new file into the database. Then the operation builds the reference hash code using the file itself and a seed for security reasons. This seed is used to provide security in the construction of the reference code.

After that the operation saves the file metadata in the database and relates the file with its metadata. Also updates the information of the file format and the file content.

The file is receiving like a "multipart/form-data" type in one servlet request.

#### Input
- **Name**: String. It corresponds with the file name.
- **experimentID**: int. It corresponds with the experiment identification.
- **serviceID**: int. It corresponds with the service identification.
- **Format**: String. It corresponds with the file format.
- **Content**: String. It corresponds with the file content annotation.

#### Output:
- **Reference code** if OK.
- **0** if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/DicodeStorage/addFile/{name}/{experimentID}/{serviceID}/{format}/{content}
```

<table>
<thead>
<tr>
<th><strong>Interface</strong></th>
<th><strong>Operation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DicodeStorage</strong></td>
<td><strong>addURI</strong></td>
</tr>
</tbody>
</table>

This operation adds a new file accessible in external location for the system. For that, the service saves the file URI in the database and then the operation builds a reference hash code using the file URI and a seed for security reasons.
As *addFile* operation, *addURI* saves the file metadata in the database and relates the file with its metadata. It also updates the information of the file format and the file content.

**Input:**
- **Name:** String. It corresponds with the file name.
- **URI:** String. It corresponds with the file URI.
- **experimentID:** Int. It corresponds with the experiment identification.
- **serviceID:** Int. It corresponds with the service identification.
- **Format:** String. It corresponds with the file format.
- **Content:** String. It corresponds with the file annotation content.

**Output:**
- **Reference code** if OK.
- **0** if an error occurred.

The output of the service is in the format “text/plain”.

The way to invocate this operation is as follows:

```plaintext
http://${server}:${port}/DicodeStorage/addURI/{name}/{URI}/{experimentID}/{serviceID}/{Format}/{Content}
```

### getFile

This operation recovers one file from the database. It distinguishes if the file is stored in the database or is accessible through an URI.

In the first case the operation recovers the file indicated by the reference code and returns the file to the user who asked for it.

In the second case the operation recovers the file using the file’s location URI and then returns the file to the user who asked for it.

**Input:**
- **Reference:** String. It corresponds with the file reference code.

**Output:**
- **1** and the **file** if OK. This file is returned as "multipart/form-data" in a response message.
- **0** if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invocate this operation is as follows:

```plaintext
http://${server}:${port}/DicodeStorage/getFile/{reference}
```

### updateMetadata

This operation updates the metadata information of the metadata identification given.

First of all checks which parameters have been requested for update.
Then proceeds to build the SQL sentence with the parameters received.

**Input:**
- **Mandatory parameters:**
  - **metadataID:** Int. It corresponds with the metadata identification.
- **Optional parameters:**
  - **Name:** String. It corresponds with the file name.
  - **Local:** byte. It corresponds with the field in the database that indicates if the file is stored in the database or is accessible through an URI.
  - **ExperimentID:** Int. It corresponds with the experiment identification.
  - **ServiceID:** Int. It corresponds with the service identification.
  - **CreationDate:** Int. It corresponds with the file creation date.

**Output:**
- 1 if OK
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:

```
http://${server}:${port}/DicodeStorage/updateMetadata/{metadataID}?name={name}&local={local}&URI={URI}&experimentID={experimentID}&serviceID={serviceID}&creationDate={creationDate}
```

**updateFile**

This operation stores the new file version in the database. It doesn’t delete the old version. This operation implements a very simple version controller of the stored files in the system. For this reason this operation links the old version with the new version of the file.

Then the operation generates the new reference code with the new file and updates the content information. Also this operation takes the remaining new file metadata information from the old file.

**Input:**
- **Reference:** String. It corresponds with the file reference code.
- **Content:** String. It corresponds with the file content annotation.

**Output:**
- **New Reference code** if OK.
- 0 if an error occurred.

The output of the operation is in the format “text/plain”.

The way to invoke this operation is as follows:
**Example usage**

User wants to upload a PDF file to the storage service. S/he clicks on the option of “Upload new file...” and completes the form presented by the system. Then, s/he selects the file from his/her local hard disk, clicks on the upload button and the file is uploaded to the storage service.

**Comments**

The Rest Services for Dicode Data Base access have been implemented using the open source framework Jersey. Jersey is the production quality, JAX-RS (JSR 311) Reference Implementation for building RESTful Web services.

Status of implementation: **in development**

<table>
<thead>
<tr>
<th>Conformance classes</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation rules</td>
<td>N/A</td>
</tr>
<tr>
<td>UML model</td>
<td>N/A</td>
</tr>
</tbody>
</table>
4 Dicode workbench user manual

In this section, the user manual of the Dicode workbench is presented. It is aimed to guide end-users in using the workbench. All available pages of the Dicode workbench are described in detail, indicating the options that users have available at each point.

4.1 Main page

Figure 4.1 shows the Dicode workbench main page.

![Dicode workbench main page](image)

Figure 4.1: Dicode workbench main page

This screen shows the welcome message. In this screen, users can create an account (2), remember their password (3) and login into the system (1).

4.1.1 Create an account

For creating an account users must click in the “Sign up” link in the main. The system then presents the page shown in Figure 4.2.
In this screen the user must fill in the empty fields with personal information and then click on the Register button (1). After that the system will send an email to the specified email address. The red asterisk indicates the fields which are mandatory fields. Also, the user has the possibility of deleting all fields at any moment with the Empty Fields button (2). If all user information is in the correct format the system shows the next screen:

Figure 4.2: Create an account screen

Figure 4.3: Create an account process success screen
This screen indicates that the registration has gone well. The user can go back clicking in the “Go back” link. Additionally, an email is sent to the user for the final account activation. The email looks like follows:

**Figure 4.4:** Activation mail sent to the users

When the user clicks on the link sent the account is activated. On the other hand, if the new email or username are already registered in the system, the next screen will be shown:

**Figure 4.5:** Account activation mail sent to users

In this screen the system indicates that the user name and the email are already registered in the system. Red messages show this information.
4.1.2 Remember password

In case the user wants to remember his/her password, s/he must click on the “Forgot password?” link on the system main page. After that the system will show the next screen:

![Figure 4.6: Remember password page](image)

Here the user needs to introduce his email in the email address field and then click on the submit button. If the email is already registered, the system will show the next page:

![Figure 4.7: Success page for the remember password process](image)
Figure 4.7 indicates that the system will send an email to the specified email address with a new password. This password must be changed by the user in the next login.

In case the email is not registered, the system will show the following page:

![Figure 4.8: Error page for the remember password process](image)

### 4.2 Login into the system

To login into the system, the user has to fill in the user and password fields in the login box. If username and password are not correct, the following page is shown:

![Figure 4.9: Error page when user types an incorrect username or password](image)
4.2.1 Welcome page

Once the user has logged in successfully, the system presents the following page:

A menu box is displayed on the upper left side, and a welcome message is shown on the right side. The users can go to their profile (2), see services in the system (3), workspaces (4), general information (5) and the frequently asked questions (6). Also they can click on the welcome link (1) to return to this page anytime. The users can also logout of the system by clicking the Exit link (7).

4.2.2 Profile section

In this page (see Figure 4.11), users can update their personal data, such as full name (1), email (2) and organization (3). To do that, they must fill in the fields with the new information and then click the “Save changes” button (7).
Also users can change their password. To do that, users must type their current password (4) and the new password twice (5 and 6), and then click on the save changes button (8). If the current password is incorrect, the system shows the next page:

Figure 4.11: Profile section page

Figure 4.12: Invalid password page
The user has to write the new password twice. If one of these two times the new password doesn’t match with the other, the system presents the next page:

![Image](image_url)

**Figure 4.13:** Page shown when passwords typed by the user do not match

If the user completes all fields properly, a success page is shown as presented in Figure 4.14.

![Image](image_url)

**Figure 4.14:** Success page for password update
4.2.3 Services

This page presents a list of the services published into the Dicode workbench.

Figure 4.15: Page showing the list of services published in the Dicode workbench

In this page the user can see the service names, descriptions, types, who published the service and the date in which the service was published. The user can see the detailed information for one service by clicking on the service name, as depicted in Figure 4.16.

Figure 4.16: Details of services within the Dicode workbench
The information about the services presented in Figure 4.16 includes: (1) Name of the service specified by the publisher, (2) invocation method, (3) URI, (4) description of the service, (5) creation date, i.e. when the service was published in the workbench, and (6) the last modification date.

### 4.2.3.1 Publish a new service

From the page shown in Figure 4.15, service providers may select the option of “Publish a new service…” to carry out this action. Clicking on this link, the system presents the page of Figure 4.17 to users.

![Figure 4.17: Publish a new service page](image)

For publishing a service, first, the user must choose the type of service (1). The system gives the user two basic options: REST and SOAP. Whether the user wants to publish another type of service, s/he should select the option “Other”. Then, some extra information has to be specified, such as the name of the service (2), textual description of the service (4), invocation method (10) or the service category (3). At the moment, service categories are limited to three options: acquisition, processing and visualization.

To facilitate location and searches of services, they can be semantically annotated using concepts of the Dicode ONtology (DON). The sensemaking operations field (5) is intended for this purpose. Users can select several operations from this list to annotate services.
Optionally, users may upload an XML description file of the service (6). Finally, the URI where the service is deployed has to be established (7).

For more complex services, the system allows users to specify parameters (8) and return types (9). At the beginning, no parameters or return types are displayed. The system allows users to add all necessary parameters and return types. Whenever the user adds a new parameter or return type, the web form is automatically expanded as presented in Figure 4.18.

![Figure 4.18: Expanded page to add parameters and return types during the service publication](image)

The “Add new param” button (1) allows the user to add a new parameter associated to the service. For each parameter, user must specify the parameter name (2), parameter type (3) and parameter data type (4). The system allows removing parameters just clicking on the “x” button (5) placed on the right of each parameter.

Additionally, users must specify the return types of the service. The user can add a new return type to the service by clicking on the “Add new return type” button (6). Then, the user must introduce the return type name (7) and the data type of the return type (8). As parameters, return types can be removed (9).

Finally, users can choose between publishing the new service, by clicking in the “Publish” button (10), or resetting the information by clicking on the “Reset” button (11). Whenever an error situation arises during the publication process, an error message is displayed in red on the page. Figure 4.19 shows an example of the error message that the user receives when s/he tries to publish a new service without specifying a name and a URI.
4.2.4 Workspaces

Clicking on the “workspaces” option from the menu box, users obtain the list of the workspaces available in the system. This list is presented as in Figure 4.20.

![Figure 4.20: List of workspaces stored in the Dicode workbench](image)

This page shows the name of the workspaces, a brief description, its visibility (public or restricted), the name of the user who created the workspace, and the date of creation. Users can create new workspaces by clicking on the link “Create a new workspace” (1). By clicking the name of a workspace, the user is moved to the associated collaborative environment. A generic view of one workspace is presented in Figure 4.21.
Figure 4.21: Collaborative workspace of the Dicodex workbench

On the left side, the widgets containing the services selected by the users of the workspace are shown. The collaborative environment is displayed in the middle of the page, providing three different views (mind-map, forum and argumentation) of the same contents. On the right side, there are two widgets: (upper) a widget to look for services and (lower) a widget with the information about the workspace.

4.2.4.1 Create a new workspace

When users want to create a new collaborative workspace, the system presents the page shown in Figure 4.22. This page contains a form with different fields that must be completed by the user. These fields are: (1) the name of the workspace, (2) a textual description of the workspace, (3) a list of tags or keywords separated by commas to annotate the workspace, (4) the domain that the workspace belongs to, and (5) the visibility of the workspace: public or restricted. Whenever the form is completed, users can create the new workspace just clicking on the “Create workspace” button (6).
As mentioned before, the visibility of workspaces can be public or restricted. Public means that any registered user in the Dicode workbench is allowed to access to the contents of the workspace. To prevent from unauthorized accesses, the visibility of workspaces can be established to “restricted”. Then, the page shown in Figure 4.23 is presented to the user, allowing him/her to select the users that will be able to access to the workspace.

Figure 4.22: Page for creating a new collaborative workspace

Figure 4.23: Defining a workspace for a restricted list of users
When user selects the “Restricted” option (1), two lists containing user names are displayed below. The list on the left (2) presents the users granted to access the workspace, and the list on the right (3) shows the names of the users within the Dicode workbench. To move user names from one list to another, user must use the buttons (4 & 5) which appears between both lists. Note that, whether the workspace visibility is established to “restricted” and no users are added to the list of granted users, the workspace will be only available for the workspace creator.

Finally, whenever an error occurs, some error messages are displayed in red at the bottom of the page. For instance, Figure 4.24 shows the error that appears when name or domain field are empty.

![Figure 4.24: Example of an error during the workspace creation](image)

### 4.2.4.2 Adding new services to the workspace

Within the Dicode workbench, workspaces can be customized by the users with different services. To add a new service, users have to use the “service search” widget located on the upper right side of the workspace.

Figure 4.25 (a) shows a basic search form with the following options: (1) name, (2) type, and (3) category of the service. Additionally, some extra filters (4) are available to refine searches. When the user unfolds the filters (4), the search box is expanded as presented in Figure 4.25 (b), allowing users to refine searches by data types returned by services (2) and by publishers of services (3).

When the user launches the search (5), the list of matches found is displayed as shown in Figure 4.25 (c). The system presents the total of matches (1), and for each service, it is shown the service name (2) and a link (“+ Use”) to add the service to the current workspace (3).
4.2.5 General Information

As shown in Figure 4.26, this page presents general information about the Dicode project, such as a description of the goals and objectives (1), a description of the three initial use cases (2), the list of partners involved (3) with links to their official web pages, and contact information to the project coordinator (4).
4.2.6 Frequently Asked Questions (FAQs)

This section is aimed to compile questions and answers to problems that users usually have using the Dicode workbench. This page looks like the one presented in Figure 4.27.
Figure 4.27: Frequently asked questions (FAQ) page
5 Usage examples

In this section, we present several scenarios focused in the three use cases of the Dicode project. Some scenarios have been redefined according to the comments received during the 1st Technical Annual Review, but others have been directly taken from the specifications presented in Deliverable D2.2: “The Dicode approach: User requirements, conceptual integrative architecture, agile methodology and functional specifications”

For each scenario, two elements are presented: (i) a textual description of a typical scenario showing how professionals act in such situations, and (ii) the envisioned workspace that Dicode will provide to those professionals and how they will use the Dicode workbench to carry out their work. Currently, a limited number of services have been integrated in the Dicode workbench. Hence, figures presented represent the future configuration that workspaces will have.

5.1 Use case 1

5.1.1 Scenario 1 (Linking genomic data to proteomic/gene ontology (GO) data)

Two researchers, Paul (statistician) and Charlotte (bioinformatician), want to expand their genomic data analysis, which they have conducted following steps 1-6 in D2.2 Table 4, and consider integrating proteomic and gene ontology data (step 1). They both agree that that would augment the robustness of the genomic analysis results (step 6), although heterogeneous data integration often is found to produce sensitive results depending on the integration method (steps 4-5). Charlotte supports the previous statement by supplying a list of PubMed IDs in a CSV file, describing recent integration methodologies (issues to consider in steps 1-2).

Paul is already aware of some of the publications suggested by Charlotte, and argues that before deciding on the integration methodology, they should further investigate the properties of their statistically significant gene lists (step 5). Paul adds that it will be interesting to see how their significant genes map to protein identifiers and GO identifiers (step 4). Tom (biologist) drops in the discussion to agree with Paul’s post adding that in the case of only very few mapped identifiers, Paul and Charlotte should consider extra datasets for their initial analysis (steps 1-2). Paul and Charlotte agree with Tom, and Tom suggests some tools he has already used for similar mapping (step 4):

- Clone|Gene IDConverter
- GeneMerge
- High-Throughput GoMiner
- ProteinOn

Also, Tom argues that a text-mining tool, such as GeneMerge, would help them to focus on only those publications/datasets referring to interesting, for their research, GO terms (step 1). Paul and Charlotte have both previously used the Clone|Gene IDConverter, whereas Paul also suggests two R/ Bioconductor packages for mapping:

http://dicode-project.eu/sites/default/files/D2.2-TheDicodeApproach-v6-EC_0.pdf
1) goTools
2) biomaRt (or the stand-alone tool BioMart)

Paul thinks GO text-mining is a good idea and argues that it can be easily performed using R (e.g. tm package) since it is always more handy to keep one file of code given that their initial analysis (steps 2-6) is in R.

Charlotte, who has also used the two R/Bioconductor packages suggested by Paul, says that she would too prefer the Bioconductor choice. Charlotte adds that the Clone|Gene IDConverter is only a mapping tool, whereas it would be more useful for them to have some measures of significance for each of the mapped terms (step 4, but can be also inferred in step 6); for instance, there are software that rank their results based on statistical tests, such as the hypergeometric test and the Fisher’s exact test. Tom explains that the three last tools in his list offer a p-value for each of the GO terms, and Paul adds that R, being a statistical package, can also do these tests.

Charlotte agrees with Paul; she thinks that both of the R packages are useful, however biomaRt is what they should look at first, since it offers the choice of retrieving, for example, proteins, SNPs and mouse homologs mapping data (steps 1-3), which is their main interest. She uploads an R script (see Figure 5.1) with some biomaRt functions to show Paul and Tom what she has in mind. Finally, they all agree they should try (at least) five different tools to cross-validate their mapping results (step 4) and have two options, i.e. union and intersection of the (at least) five outcomes, to further their analysis (steps 5-6).

```r
library(biomaRt)

mart <- useMart(biomart="ensembl", dataset="hsapiens_gene_ensembl")

genes <- read.csv("significant_genes_file.csv")

results <- getBM(attributes = c("ensembl_gene_id", "hgnc_symbol"),
 filters = "hgnc_symbol", values = c(rownames(genes)), mart = mart)

m<- match(rownames(genes),results$hgnc_symbol)

genes$hgnc <- results[m,"hgnc_symbol"]
genes
```

**Figure 5.1:** Example of an R script code used by the researchers

5.1.2 Envisioned workspace for UC1_Scenario 1

Figure 5.2 presents the envisioned configuration of the Dicode workspace for scenario 1 of Use Case 1. At the beginning, in the middle there would be the collaborative service and no services would be displayed on the left.

With this configuration, discussions between researchers would be conducted by using the collaboration service (middle widget). The collaboration service (described in section 3.2.2) would help researchers to decide which tools they should use. After discussion, researchers would decide to execute R scripts. Therefore, researchers would look for some service to store the scripts and some services to execute those scripts. Using the “service search” box, they
locate a “Storage service” and a service to execute R scripts. Both services are added to the workspace. Then, Charlotte would upload the R script to the Storage service. From this moment, the R script would be available for the rest of the users participating in the experiment.

After that, to execute the R script, any user could click on the file displayed in the Storage service and drag it into the “R engine” box. This action will load the script in the “R engine” widget and it could be executed. The results of the execution would be automatically stored in the Storage service. These results could be analyzed by any of the users and new discussions or conclusions could be carried out.

![Image of DICODE workspace](image_link)

Figure 5.2: Envisioned workspace for Use Case 1 – Scenario 1

5.2 Use case 2

5.2.1 Scenario 1

(note: This scenario has been directly extracted from Section 4 of D2.2)

Consider a patient Joan that has problems with her joints and wrists.

1) Radiographer Alice takes the scans of a patient’s wrists and joints checked the anatomy position and the quality of the images captured.

2) Radiologist Chris is notified by the Radiology Information System (RIS) that a new case can be seen in the PACS. He accessed the case images through PACS.

3) Radiologist analyses images in Dynamika. Chris uploads scan images into Dynamika. Dynamika shows calculates and displays overlaid coloured maps of the tissues were shown in the dynamic scans. Using Dynamika, Radiologist performs motion. Using Dynamika, Chris identifies regions of interest (ROI) which will be used together with the maps in
generating statistics about the ROIs. When doing so, Chris also made comments for his analysis. Chris stores the maps, ROIs and associated statistics in a report associated with the specific scan image. This process requires few minutes to be completed. Chris then sends the electronic form report (HTML file) to the responsible clinician, David.

4) David receives the report from Chris. He uploads the images to Dynamika and repeats the steps performed earlier by the radiologist: identifying ROIs and generating maps and statistics. Finally he stores the maps, ROIs and associated statistics in a report, (which might be different from the one the radiologist produced earlier), associated with the specific scan image.

5) David then consults previous relevant statistics along with treatment information from previous visits of the patient, Joan. The information includes Joan’s journal and historical reference data via the Patient Trial Details. He further examines the trial protocol used at the time the scans were taken through the Trial Details, in order to reach a conclusive decision on the effectiveness of the treatment for Joan. If the images and clinical findings do not correspond to the radiologist conclusion on the specific patient then this patient’s case must be discussed further on a conference.

6) In this instance, if David has some difficulties in concluding on the effectiveness of the treatment on the specific patient. He might want to consult his colleagues, the radiologist and another clinician, Frank who are located elsewhere, to exchange ideas and collaboratively reach a decision. At present, there is no easy way to share data in real time for joint discussion.

7) Follow up analysis applies to analysis for the whole trial including multiple patients or even multiple trial sites. The output reports for each patient are also stored in CSV and imported into Excel to aggregate the results. The aggregated data then provides visibility over time and across multiple patients to reveal general trends in conditions. As an overview, this shows the success of particular treatments.”
5.2.2 Envisioned workspace for UC2_Scenario 1

![Envisioned workspace for UC2_Scenario 1](image)

**Figure 5.3:** Envisioned workspace for Use Case 2 – Scenario 1

Figure 5.3 presents the envisioned configuration of the Dicode workspace for scenario 1 of Use Case 2. In this scenario, researchers need to share Dynamika reports. As previously described, one of the users would look for a Storage service and would add it to the workspace. It would also be interesting for the users to search information about protocols or treatments in different journals (e.g. PubMed), considering the data contained in Dynamika reports. Using the “Service search” widget, researchers could find one service to analyze Dynamika reports and extract the most relevant terms (Augmentor service, described in D5.3.1: “The Dicode multi-perspective ontology engineering tool (initial version)”\(^{11}\), and another service to search in PubMed just specifying some keywords. Both services would be added and the workspace would present a similar aspect as shown in Figure 5.3.

With this environment configured, discussion between experts would be held using the collaborative service in the middle. When one expert wants to share a Dynamika report, s/he would use the “Storage service” to upload the report. Then, the report would be available for all users involved in the workspace. When one user needs to look for more information in PubMed about one concrete Dynamika report, s/he could drag the report from the “Storage service” to the “Augmentor service”. This action would invoke the “Augmentor service” analyzes the Dynamika report and would present the more relevant terms detected. Some of those terms could be dragged to the “PubMed service” and this widget would search automatically in PubMed presenting the results. Users would see the same results and could continue with the discussions and further analysis.

\(^{11}\) [http://dicode-project.eu/sites/default/files/D5.3.1-ontologyEngineeringTool-v6-EC.pdf](http://dicode-project.eu/sites/default/files/D5.3.1-ontologyEngineeringTool-v6-EC.pdf)
5.3 Use case 3

5.3.1 Scenario 1

(note: This scenario has been directly extracted from Section 5 of D2.2)

1) Kate and John start this research with finding, aggregating and summarizing information about the product of their client. She uses monitoring tools to see what public opinion is from the Internet, especially from social media web sites, such as Facebook and Twitter. There are many monitoring tools available, some also free of charge. To get better result, she has to use several tools at the same time and summarize the information from all the tools she uses.

This step is an interactive activity because of the limitation of the tools and the noise of the data retrieved by the tools. Knowing that there are many noises in the retrieved information, Kate has to browse through and uses her judgment to pick out the relevant posts or conversations.

2) With the relevant posts and conversations, Kate has to analyze it to meet her research goal. In this case, she would like to find out:

- Key trends (What are the conversations about?)
- Brand Performance (How is the brand/service/product being discussed?)
- Competitive Benchmarking (How do you compare with competitors?)
- Key influencers (Who is shaping the conversations?)
- Overtime performance (How does the brand perform over time?)

There are different methods to conduct this kind of analysis. One method is called ‘sentiment analysis’ – that is splitting the information into positive, neutral and negative comments. Looking at the content of positive posts and negative posts, further analysis is conducted to see if there are any emerging themes coming from the posts (e.g. by clustering of opinions), and to judge how influential those opinions would be. The latter requires:

- Knowledge of the analyst (e.g. one would judge that opinion from the newspaper Independent on ‘politics’ will be more influential than their opinion on, say, ‘beauty’), and
- Tracking how often the opinion is cited elsewhere. Tools for ‘influential analysis’ exist (e.g. Linkfluence’s Engage package) but rather static pictures of affairs were presented.

The outcome of the analysis will be used, in conjunction with other input, in the preparation for the recommendation.

5.3.2 Envisioned workspace for UC3_Scenario 1

Figure 5.4 presents the envisioned configuration of the Dicode workspace for scenario 1 of Use Case 3. In this case, users need to collect information from social media, analyze them and finally graphically presenting the results. First, users would look for services aimed to collect information from social media. For instance, a “Twitter harvester” (described in D3.1.1: “The Dicode Data Mining Framework (initial version)”12) service could be found and added to the workspace. Users would be able to configure such service to specify the terms or topics of interests, size of the sample to retrieve and other parameters.

---

Figure 5.4: Envisioned workspace for Use Case 3 – Scenario 1

Then, users could discuss, using the collaboration service in the middle, the kind of visualization which they prefer to present the results. They would look for visualization services and would select a couple of services presenting data as a pie chart and as a line chart. To represent any of the datasets harvested by the “Twitter harvester”, users could drag that dataset into the visualization services and automatically, the data will be presented as shown in Figure 5.4. Looking at those results, users could take decisions or discuss further.
6 Conclusions and Future Work

In this deliverable, we have presented work carried out during the first fifteen months of the project concerning the integration of Dicode services. Deliverable D5.4.1 gathers information coming from Tasks 5.3 and 5.4.

We introduced the Dicode workbench which has been implemented as a dynamic web application. A general description, site map and envisioned user roles has been presented, together with a complete user manual. Furthermore, the integration strategy selected by Dicode consortium has been introduced. This strategy is based on three main pillars:

- **Widget-based user interface.** This approach has been selected to create a flexible and customizable web user interface. Each widget will present one different service to the users. Interactions between widgets are being investigated and will be allowed in future versions of the Dicode workbench.

- **Dicode Registry of Service (DRS).** This registry will contain all the information about services to ensure interoperability between them.

- **Single Sign-On strategy.** Some service will require authentication from its users. The Dicode workbench will simplify this process to users, asking just once for the credentials when users login into the system.

A first functional prototype of the Dicode workbench has been deployed at [http://hodgkin.dia.fi.upm.es:8080/dicode](http://hodgkin.dia.fi.upm.es:8080/dicode) for testing purposes and internal evaluation. At the moment, this prototype integrates most of the services presented in the deliverable. The services successfully integrated at the moment are:

- User management service
- Service management service
- Workspace management service
- Collaboration support services

Finally, representative usage examples have been depicted focused in the three use cases of the Dicode project. For each usage example, we presented a typical scenario based on the specifications established in deliverable D2.2. We showed the envisioned workspaces within the Dicode workbench in which users will carry out their tasks in a flexible way and in collaboration with other users. In the coming months, we plan to finish the development of more services (data mining, collaboration, management, etc.) and integrate them within the Dicode workbench following the approach described in this deliverable.
References


