Wf4Ever: Advanced Workflow Preservation Technologies for Enhanced Science

STREP FP7-ICT-2007-6 270192

Objective: ICT-2009.4.1b — "Advanced preservation scenarios"

D1.2v1 Wf4Ever Sandbox v1

Deliverable Co-ordinator: **Raul Palma, Marcin Werla**

Deliverable Co-ordinating Institution: **PSNC**

Other Authors: **Michał Nowak (PSNC), Mateusz Matela (PSNC), Piotr Hołubowicz (PSNC), Stian Soiland-Reyes (UNIMAN), Jiten Bhagat (UNIMAN), David De Roure (OXF), Graham Klyne (OXF)**

This deliverable presents the Wf4Ever Sandbox with the results of the first iteration on the review and deployment of software relevant for the development of components or applications based on scientific workflows.

Document Identifier:	Wf4Ever/2010/D1.2v1/v1.0	Date due:	May 31, 2011
Class Deliverable:	Wf4Ever FP7-ICT-2007-6 270192	Submission date:	May 31, 2011
Project start date	December 1, 2010	Version:	v1.0
Project duration:	3 years	State:	Final
		Distribution:	Public

Wf4Ever Consortium

This document is part of the Wf4Ever research project funded by the IST Programme of the Commission of the European Communities by the grant number FP7-ICT-2007-6 270192. The following partners are involved in the project:

Intelligent Software Components S.A. (ISOCO) – Coordinator Edificio Testa, Avda. del Partenón 16-18, 1^o, 7^a Campo de las Naciones, 28042 Madrid Spain Contact person: Jose Manuel Gómez Pérez E-mail address: jmgomez@isoco.com	**University of Manchester (UNIMAN)** School of Computer Science Oxford Road, Manchester M13 9PL United Kingdom Contact person: Carole Goble E-mail address: carole.goble@manchester.ac.uk
Universidad Politécnica de Madrid (UPM) Departamento de Inteligencia Artificial, Facultad de Informática. 28660 Boadilla del Monte. Madrid Spain Contact person: Oscar Corcho E-mail address: ocorcho@fi.upm.es	**Instytut Chemii Bioorganicznej PAN - Poznan Supercomputing and Netowrking Center (PSNC)** Network Services Department Ul Z. Noskowskiego 12-14 61704 Poznań Poland Contact person: Raul Palma E-mail address: rpalma@man.poznan.pl
University of Oxford (OXF) Department of Zoology South Parks Road, Oxford OX1 3PS United Kingdom Contact person: Jun Zhao, David De Roure E-mail address: jun.zhao@zoo.ox.ac.uk david.deroure@oerc.ox.ac.uk	**Instituto de Astrofísica de Andalucía (IAA)** Dpto. Astronomía Extragaláctica. Glorieta de la Astronomía s/n, 18008 Granada Spain Contact person: Lourdes Verdes-Montenegro E-mail address: lourdes@iaa.es
Leiden University Medical Centre (LUMC) Department of Human Genetics Albinusdreef 2, 2333 ZA Leiden The Netherlands Contact person: Marco Roos E-mail address: M.Roos1@uva.nl	

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to the writing of this document or its parts:

- PSNC
- OXF
- UNIMAN
- UPM
- ISOCO

Change Log

Version	Date	Amended by	Changes
0.1	15-04-2011	Raul Palma	Created Document
0.2	29-04-2010	Raul Palma	Update on the table of content. Input for software section
0.3	09-05-2010	Raul Palma	Executive summary. Input for software section from wiki
0.4	11-05-2010	Raul Palma	Introduction. Roadmap input
0.45	11-05-2010	Stian Soiland-Reye	Software input
0.5	13-05-2010	Raul Palma	Conclusions
0.6	13-05-2010	Graham Klyne	Tests input
0.7	13-05-2010	Piotr Hołubowicz	Tests input and Appendixes
0.8	19-05-2010	Marcin Werla	dLibra description. Upgrade procedure
0.9	19-05-2010	Piotr Hołubowicz	Walkthrough
0.95	20-05-2010	David De Roure	Input software section
0.96	20-05-2010	Marcin Werla	dLibra description update
0.98	20-05-2010	Piotr Hołubowicz	Document updates
0.99	20-05-2010	Raul Palma	Final Modifications
1.0	30-05-2010	Raul Palma	QA comments addressed

Executive Summary

This document presents the W4Ever Sandbox, a testing environment for developers and users, which shows the results of the first iteration on the review and deployment of software relevant for the development of components and applications based on scientific workflows. A complete description of the running environment and the software deployed is provided. In particular, we focus on the first prototype that has been developed in Wf4Ever for an early interaction with Research Objects. Additionally, a brief discussion about the role of the sandbox and its relation to the architecture and reference implementation of Wf4Ever is presented.

Two usage scenarios of the sandbox are supported. One the one hand, users are able to interact with the sandbox through a live demo site. On the other hand, users are able to download and run locally a copy of the sandbox. Hence, this document also provides a walkthrough of the sandbox, providing a guide to the users for each of the supported usage scenarios.

Finally, it is important to note that the sandbox is a very dynamic system, which will be changing over time. The software developed will be evolving and new software will be deployed as part of the continuous software review task running throughout the project life. Hence, this document presents a detailed plan for the maintenance of the sandbox over time, including its documentation and the envisioned upgrade procedures.

Contents

List of Tables

List of Figures

1 Introduction

One of the objectives of Wf4Ever is to select and deploy workflow management, social network and digital library technologies that will support the development of a reference implementation enabling the preservation and efficient retrieval of scientific workflows across a range of domains. In order to do so, we have installed a test infrastructure, known as the Sandbox, for component and application development, that can be used to evaluate the integration of partner's components with the deployed technology.

Moreover, the Sandbox supports the deployment of integrated prototypes, checked with continuous evaluations, which will integrate the services developed in WPs 2 to 4 with other technologies selected in the test infrastructure. The development of such prototypes enables to iteratively and incrementally develop a reference implementation through the continuous evolution of requirements and solutions, and at the same time, it stimulates the collaboration between the project teams. Moreover, such prototypes allow for a quick response in the face of changes in software requirements and evolving research, leveraging user expertise by incorporating them throughout the development process.

This document describes the results of the first iteration on the review and deployment of software in the Wf4Ever Sandbox, which is relevant for the development of components or applications based on scientific workflows, their sharing, within social networks, semantic technologies, and digital libraries. The Sandbox has been configured as a virtual machine (VM) that is publicly available, and it can be used either via a live demo site or by downloading and installing locally a copy of the VM disk image. Besides from a detailed description of the running environment and software deployed in the Sandbox, this document also provides a guide on how to use both the demo site and the VM, and an overview of the update procedures. A major component of the Sandbox is the first prototype that has been developed in Wf4Ever project. This prototype serves as an experimental pilot providing exemplar reference services and forming a test infrastructure that can be used by all partners.

1.1 Sandbox Role

The Sandbox plays an important role within Wf4Ever project. It provides a testing environment, a sort of playground, for users and developers so that they can experience and interact with Wf4Ever technologies and related software as early as possible during the development process. Moreover, it is the infrastructure that will be used to integrate existing technologies so that the rest of WPs can perform tests, and where the final reference implementation will be deployed.

For users, for example, the Sandbox provides them the possibility to try and test the software developed, enabling them to comment and provide timely feedback and ensuring that their requirements are continuously taken into account during the development of the reference implementation. For developers, the Sandbox offers a shared infrastructure to test that the software developed works as expected with the rest of the technologies to be integrated, allowing them to identify potential risks and problems.

Finally, the software deployed in the Sandbox enables users to interact with exemplary objects, e.g., research objects, and other sample resources being specified as part of the research work during the project lifetime. Again, this will enable technical WPs to take user's requirements into account throughout all the work phases.

For example, after the first iteration of the implementation of the first prototype, users from WP5 and WP6 had the opportunity for the first time to play around the project software and start creating some toy ROs[1]. Their experience and feedback was highly valuable during the next iteration of the prototype development.

1.2 Relations with other WPs

As explained above, the Sandbox includes both services and software components (e.g., prototypes) that are developed within the project, as well as selected software relevant to the development of our reference

[1] See the outcome of the demonstration at `http://www.wf4ever-project.org/wiki/display/docs/2011-04-11+Demo+to+Users`

implementation.

Hence, the Sandbox, and in particular the software deployed, highly depends on the work carried out in the other WPs. On the one hand, the services and software components developed by technical WPs (2-4) will be integrated and tested in the Sandbox. Similarly, the workflows for astronomy and genomics implemented by the use case WPs (5-6) will be used to create sample data that will be available in the Sandbox to allow users the construction and interaction with ROs around those communities.

On the other hand, during the selection of the software to be deployed in the Sandbox, we take into consideration the expected benefits of such software for Wf4Ever components from the technical WPs as well as the expected applications from our users in WP5 and WP6.

Moreover, the software depends on and implements the models and concepts that are being specified as part of the research work carried out in the technical WPs. For example, for the development of the first prototype, a draft model of a Research Object was required from WP2 in order to implement some basic services around those objects (e.g., store and retrieve). In this case, the ADMIRAL data package model was provided as input (see Section 2.2.2).

The remainder of this document is organized as follows: Section 2 describes the Sandbox configuration and software deployed. In particular, an overview of the running environment and a detailed explanation of the first prototype is presented, followed by a description of the tests performed on the prototype and a brief discussion regarding the relation of the Sandbox software with the Wf4Ever architecture and reference implementation. Section 3 provides a walkthrough the Sandbox, both for the live demo site and for the VM image. Next, in Section 4, an outline of the Sandbox maintenance plan is presented, in particular regarding the maintenance of the documentation and upgrade procedures. Finally, after the conclusions in Section 5, technical details regarding the REST interfaces are presented in Appendix A and regarding the development and deployment instructions of the prototype are presented in the Appendix B.

2 Sandbox Software

The preparation of the Sandbox consisted mainly in the selection of the software to be used and deployed. Such software includes not only the software developed or selected for its relevance towards a workflow preservations system, but also the software of the running environment where the applications will be running. Additionally, some more technical decisions, regarding where it will be located and how it will be accessed and used had to be made.

2.1 Configuration Overview

The Sandbox has been implemented as a VMWare[2] virtual machine (VM) that can be used in the following ways (a detailed description of how to use the Sandbox is presented in Section 3):

- A live demo site available at `sandbox.wf4ever-project.org`.
- A VM disk image that users are able to download in order to run a local copy of all the software deployed.

Moreover, the Sandbox VM has been configured with the following running environment:

- Ubuntu 10.04
- MySQL 5.1
- Java 1.6
- Git 1.7
- Ruby 1.9.2 / RubyGems 1.6

Additionally, the following software components that constitute the first prototype of Wf4Ever have been deployed:

- ROBox
- ROSRS
- dLibra

In the next section we detail the prototype rationale and implementation.

2.2 First Prototype

The first prototype is an early implementation to support the interaction (e.g., store and retrieve) of scientists with Research Objects (ROs). It handles the connection of a Dropbox account to a Wf4Ever RO service, in order to support an automated publication and synchronization of ROs between a DropBox folder and the dLibra system. Dropbox provides a powerful, well known and ready made environment for working with ROs. It is one of many possible environments we can integrate with, but one that we can use to quickly build a prototype around to then elicit user feedback and requirements and evolve the architecture.

[2] `http://www.vmware.com/`

2.2.1 Background

The first phase when a user wants to create a new RO will generally be to start populating it with actual files and resources. For a typical scientists these files would be stored across a few folders on a laptop or USB stick, while more advanced users might use a file server, Dropbox or SVN.
These files and resources can for instance be a selection of:

- Experiment Protocol (e.g., in MS Word)
- Results from previous experiments
- New experiment data (from instruments, measurements, etc.)
- Discarded experiment data (failed calibrations, etc)
- Reference data sets (downloaded from official and unofficial sources)
- Scripts/tools/workflows for processing data
- Compiled libraries and binaries needed by scripts
- Raw outputs and logs from running tools over data (unsuccessful ones hidden in a deeper folder)
- Results compiled from output (e.g., in MS Excel)
- Interpretations and summary tables (e.g., in word)
- Draft long paper for publicising results (e.g., in LaTeX)
- Referenced papers (PDFs)
- Published workshop paper (e.g., in word and PDF)
- Presentation at workshop/conference (e.g., in MS Powerpoint)
- Notes made after talking to people at workshop/conference (e.g., in OneNote, Notepad)

As users are generally confident with working with files and folders, and are increasingly using services like Dropbox to achieve replication, sharing and archiving/backup of their data, the idea behind was that Wf4Ever can tap in to the Dropbox account and make this be an interface for populating ROs. At the same time users keep working with their files pretty much as before, but with the added value of Wf4Ever tools allowing them to add further annotations, templates and guiding them towards building a fully fledged RO.

2.2.2 Research Object Structure

The structure of the Research Object that we are using in this prototype is based on the ADMIRAL data package[3].
From users' point of view, in this prototype a RO is a directory in the file system. All files included in the directory or any of its subdirectories are considered to be resources of the RO, except for the file named manifest.rdf (placed directly in the RO directory). The manifest.rdf is an RDF/XML file containing metadata associated with the RO as well as a list of all its resources. This file contains:

- descriptive metadata (it can be edited by the user):
  - in the dcterms namespace;

[3] http://www.wf4ever-project.org/wiki/display/docs/Data+packages+in+ADMIRAL

  - in the oxds namespace (ADMIRAL specific metadata);
    * oxds:currentVersion tag contains the identifier of the current version of the RO

- structural metadata in the ore namespace, should be generated automatically by the ROSRS on the basis of the contents of the RO.

As an addition to the definitions of ADMIRAL Data Package, we define the following assumptions:

- If several versions of a RO have been defined, information about all the versions will be automatically listed in manifest.rdf (in the dcterms:hasVersion tag).
- For each local file listed in the ore:aggregates tag, MD5 checksum and the last modification time will be stored as additional attributes of the tag.

The following example shows the structure of the manifest.rdf file:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:dcterms="http://purl.org/dc/terms/"
    xmlns:ore="http://www.openarchives.org/ore/terms/"
    xmlns:oxds="http://vocab.ox.ac.uk/dataset/schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 >
   <rdf:Description rdf:about="http://example.org/admiral-test/datasets/DatasetsSubDir">
     <oxds:isEmbargoed>True</oxds:isEmbargoed>
     <dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2011-02-02T14:17:52.667536
     </dcterms:modified>
     <dcterms:creator>admiral</dcterms:creator>
     <rdf:type rdf:resource="http://vocab.ox.ac.uk/dataset/schema#Grouping"/>
     <ore:aggregates
     rdf:resource="http://example.org/admiral-test/datasets/DatasetsSubDir/admiral-dataset.txt"/>
     <ore:aggregates
     rdf:resource="http://example.org/admiral-test/datasets/DatasetsSubDir/DatasetsSubDir"/>
     <ore:aggregates
     rdf:resource="http://example.org/admiral-test/datasets/DatasetsSubDir/DatasetsSubDir/s1.txt"/>
     <ore:aggregates
     rdf:resource="http://example.org/admiral-test/datasets/DatasetsSubDir/DatasetsSubDir/s2.png"/>
     <dcterms:isVersionOf
     rdf:resource="http://example.org/admiral-test/datasets/DatasetsSubDir-packed/DatasetsSubDir.zip"/>
     <dcterms:title>Title for dataset DatasetsSubDir</dcterms:title>
     <dcterms:identifier>DatasetsSubDir</dcterms:identifier>
     <dcterms:description>Description for dataset DatasetsSubDir</dcterms:description>
     <oxds:currentVersion>8</oxds:currentVersion>
     <oxds:embargoedUntil>2081-01-15T14:17:52.666320</oxds:embargoedUntil>
     <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2011-02-02T14:17:52.666744
     </dcterms:created>
   </rdf:Description>
</rdf:RDF>
```

2.2.3 Use Case Scenario

A simple scenario of this use case (as seen by the user): (*Assumption*: User already have Dropbox installed and configured on their desktop computer.)

1. User registers their Dropbox account with the prototype website.
2. User specifies a folder in Dropbox using the prototype website, which will be used to store and find ROs.
3. User creates a subfolder *Cool Experiment* within this folder.
4. Prototype recognizes the new folder as a new RO called "Cool Experiment".
5. Prototype creates a RO manifest and place it in the Dropbox as *My Research Objects/Cool Experiment/manifest.rdf* (one top-level manifest per RO).

6. User adds files and folders *to Cool Experiment*, for instance, data/measurements-2043.csv, results.xls and analysis.doc.
7. Prototype discovers the files, and adds them to the RO manifest.

The user now has created a minimal and self-contained RO. They may place this folder structure wherever they want, say on a USB stick or zipped up on their web site. The manifest will contain identifiers referring back to the service, and provides the starting point for other Wf4Ever services to start annotating the captured resources and the RO itself.

ROs will change through time, e.g., new resources will become part of the RO and some will become unnecessary. Table 1 summarizes the expected tasks to be performed by the prototype for each of the user actions.

User action	**Prototype action**
Adds some RO related file to the directory	• Stores the contents of a new file. • Updates the manifest.rdf file by adding a corresponding resource element.
Modifies some RO related files in the directory	• Stores the update contents of the file. • Updates the manifest.rdf file by updating the attributes of corresponding resource element.
Remove some RO related files in the directory	• Deletes the file from the internal storage. • Updates the manifest.rdf file by removing the corresponding resource element.
Modifies descriptive metadata in the manifest.rdf file	• If RO Version is modified (the oxds:currentVersion tag): – If provided RO Version has not been used before, prototype creates a new version of the RO - in the prototype all files and metadata from the previous version are copied and become a new version of the RO. In the shared DropBox directory the only updated file is the manifest.rdf, because of the versioning information stored in it. – If provided RO Version has been used before, prototype restores the directory's state to the given RO Version. • Updates metadata of the RO Version stored in the ROSRS, according to changes in manifest.rdf.

Table 1: User actions and corresponding Prototype tasks

One can easily imagine a few interesting extensions to this scenario:

- A UI (web) for browsing the RO, their resources and metadata.
- A UI for adding additional metadata: title, description, intended usage.
- A UI for making links between resources, such as "X was produced by Y using Z". (Rudimentary provenance).
- A template system. That is, for instance, create an RO folder *AstroExperiment X* and the service can create additional folder structures data, scripts, outputs, and automatically annotate the resources according to their placement in this structure.

- "RO-aware" software (e.g., Taverna with a Wf4Ever plugin) recognizes the manifest.rdf and updates it to provide additional information about files it is saving to the RO folder.

2.2.4 Implementation

This prototype was implemented by developing two new components, reusing our existing product dLibra, and using the external Dropbox API. An overall diagram of the prototype architecture is depicted in 1.

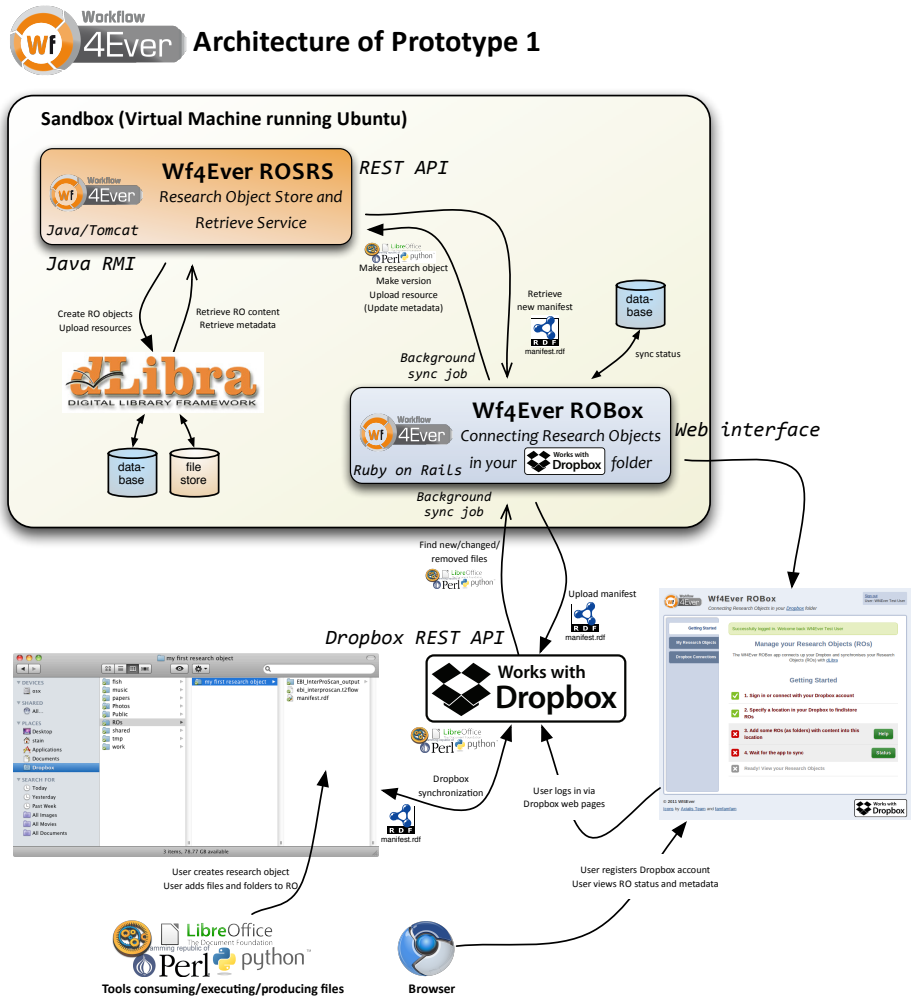

Figure 1: First prototype architecture

dLibra

dLibra (`http://dlibra.psnc.pl`) is a software system for building digital libraries. Its multitier service-oriented architecture and configurable components of end-users applications allow to deploy digital library systems suited for particular needs. The general architecture of the system is presented in 2. The core of dLibra is the following set of services:

- Metadata service - is responsible for management of the digital library objects and other elements of the digital library data model - composite objects, collections and directories; it also provides all functionality related to metadata of digital library elements including management of the metadata schema and dictionaries created for each metadata schema element.

- Content service - is responsible for management of the content of digital library objects (bytestreams); beside of store/retireve functionality provides also long-term preservation support and content transformations.
- Indexing service - is responsible for building search indexes for efficient searching in content and metadata of digital objects.
- Search service - provides searching functionality based on indexes built by indexing service.
- User service - handles user accounts and authentication/authorization operations.
- Profile service - stores private profiles of users.

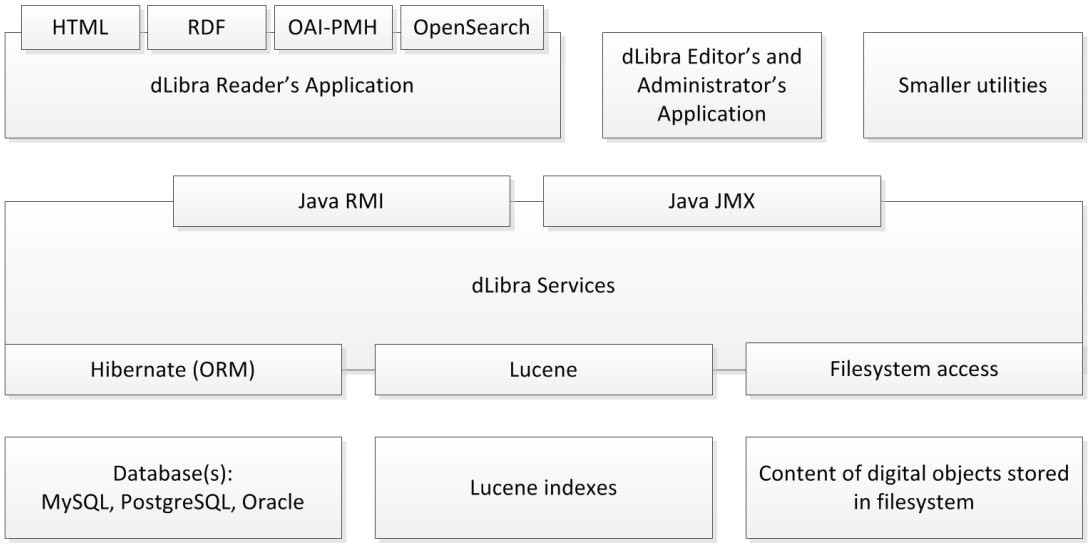

Figure 2: dLibra Architecture

In a typical dLibra configuration, end-users applications are deployed on top of those services. dLibra provides web application for readers and a set of desktop applications for editors and administrators of the digital library. However, for the prototype described in this section, instead of having some of those existing applications on top of the core dLibra services, a new one was developed - the ROSRS service described in the next section. As explained below, this service provides a REST interface to the functionality of dLibra services that is exposed via the Java RMI technology. At the moment metadata, content and user services are the most utilized ones, although indexing and searching services are also configured and can be used directly to perform search operations.

Figure 3 shows the dLibra data model. A key element of this model is the regular object, which consists of a set of files. Each file can have one or more versions. With each version a data stream is associated. Versions of files can be tied together into editions of digital object. This allows to show the changes of the digital object in its lifetime. Beside regular object, dLibra data model includes also metadata records without digital content and objects representing deleted regular objects. These three types of objects can be grouped in a composite object which can be a part of another composite object, creating a nested hierarchical structure.

Each element of the data model can be described with descriptive, administrative and technical metadata. The descriptive metadata schema, which is by default Dublin Core[4], can be customized via the Metadata Service API. The ROSRS uses this data model by mapping it to the RO structure described in 2.2.2 (see next section for more details).

[4] http://dublincore.org/

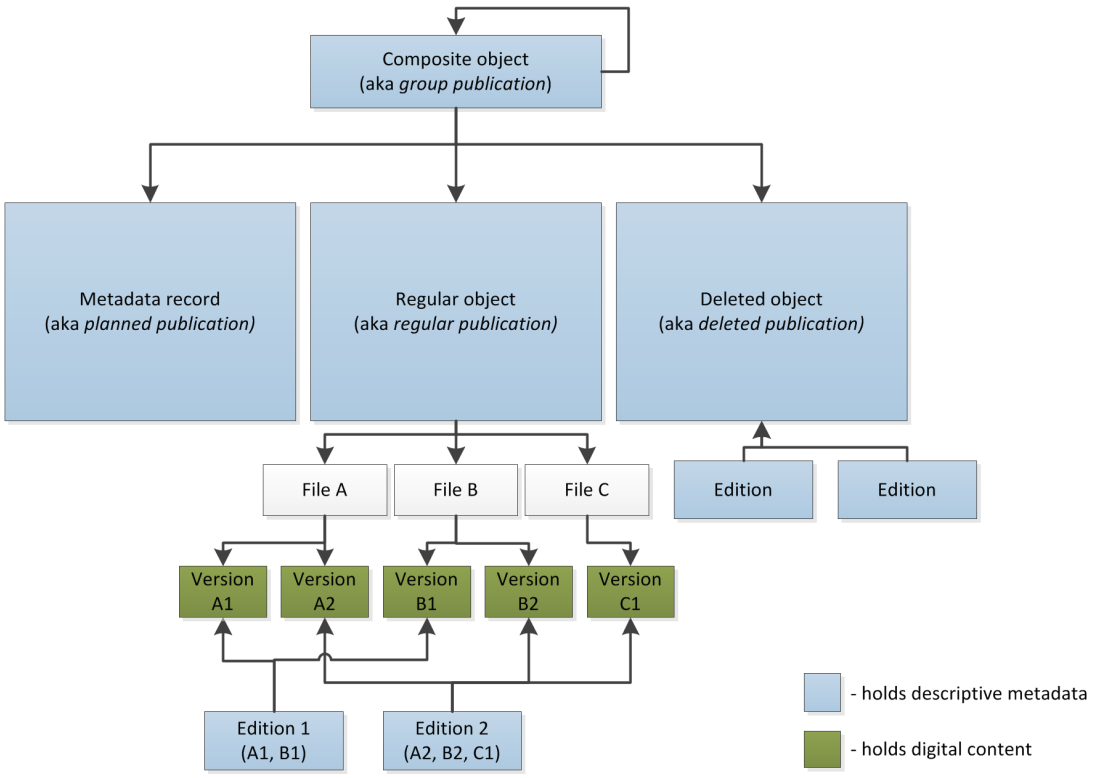

Figure 3: dLibra Data Model

ROSRS

The RO Store and Retrieve Service (ROSRS) enables the usage of a digital library for the manipulation of ROs. The service is accessible at `http://sandbox.wf4ever-project.org/rosrs2` via a REST interface (see Appendix A), and it has been implemented on top of dLibra. The service deployed corresponds to the second iteration of the implementation, which addresses the comments received from users and other technical partners after the first iteration.

As part of the service specification, we made the following design decisions:

Handling modifications in manifest.rdf file. Users are expected to modify only the tokens corresponding to the RO metadata. List of resources is automatically regenerated every time any other file in the RO directory is created, modified or removed. Therefore any changes to the resource list by users will be ignored.

Handling remote resources. Currently, the only way users can include a remote resource in the RO is by creating an internet shortcut (file with .url extension in Windows) pointing to the remote resource. The shortcut should be placed directly in the RO directory. Such link will be automatically parsed by ROSRS and the URI will be placed in the resource list in the manifest.rdf file. We need to gather additional requirements from users and check how it is handled in other systems for future versions of the service.

Handling conflicts. If two or more users edit contents of the same file at the same time, it leads to conflicts. In DropBox, this results in creation of conflicted copies - for each conflict a new file is added, with information about author and current date appended to the file name. As we have no means to resolve such conflicts, the conflicted copies will not be treated in any special way. They will be added to the ROSRS simply as separate files. As users resolve conflicts manually, it will result in removal of conflicted copies and subsequent removal of corresponding files in the prototype.

Conflicts on the manifest.rdf file require special consideration, as this file is used to keep track of changes in

metadata and resources. Furthermore, the conflict may be triggered by ROSRS itself as it updates the file contents after every change in resources. We decided not to address this issue at the moment though, in order to keep the first prototype design simple. We assume that manifest.rdf is relatively small, so DropBox will manage to synchronize it quickly between users. Also, in normal use manifest.rdf is rarely modified by users (only when metadata changes). As a result, the risk of conflicts occurring for this file is minimal.

Tables 2 and 3 summarize the mapping between the RO structure and the internal dLibra data model, and the mapping between the metadata elements in the ADMIRAL data package and dLibra, respectively.

Table 2: Mapping between RO structure and dLibra data model

RO model	dLibra model
Research Object	Group Publication
Version of Research Object	Publication with single edition. Its content is modified every time the data or metadata of the RO changes
Resource File	File. For each modification of the file a new File Version is created. The deletion of files will be handled by excluding them from the "edition" in dLibra.

Table 3: Mapping between metadata elements in ADMIRAL Data Package and dLibra

ADMIRAL Data Package	manifest.rdf	dLibra
dataset local identifier	<dcterms:identifier>	edition name
username of creator	<dcterms:creator>	edition attribute: Creator
title of the dataset	<dcterms:title>	edition attribute: Title
title of the dataset	<dcterms:title>	edition attribute: Title
textual description of the dataset	<dcterms:description>	edition attribute: Description
enumeration of the URIs of the resources in the data package	<ore:aggregates rdf:resource="http://some/url"/>	assignment of file versions to edition
version number for the data package	<oxds:currentVersion>	publication name
embargo status and date	<oxds:isEmbargoed>, <oxds:embargoedUntil>	TBD
date the package was created (submitted)	<dcterms:created>	group publication attribute: Created
date the package was last modified	<dcterms:modified>	group publication attribute: Modified
reference to any resource from which the data package has been derived	<dcterms:source>	group publication attribute: Source

Technical implementation details. The service is a servlet-based application built using the Jersey framework[5]. Additionally, Jena framework[6] is used to handle RDF files, and additional XML transformations are performed using Xalan library[7].

Development and deployment instructions. See Appendix B.

ROBox

The Dropbox RO Connector application (known as ROBox) connects up a user's Dropbox folder, containing ROs, with the ROSRS. It manages the synchronisation between the user's Dropbox folder and the backend dLibra store. ROBox is a web application accessible at `http://sandbox.wf4ever-project.org/robox`.

Overall behaviour.

- Users log in through their Dropbox accounts. The first time this is done, ROBox will store the OAuth tokens for the user's Dropbox so that background synchronisation takes place.
- Users can then register their ROs folder (see below for more details).

[5] `http://jersey.java.net/`
[6] `http://jersey.java.net/`
[7] `http://xml.apache.org/xalan-j/`

- ROBox will then continually monitor this folder for new and updated ROs and handle synchronisation - changes will be pushed to the ROSRS and updates from the ROSRS will be pulled back into the user's Dropbox folder.
- The ROSRS Client Ruby Gem (being developed as part of the ROBox work) is used to interact with the ROSRS service.
- After the first sync, a Dashboard is accessible to the user. Here the user can see the status of their Dropbox ROs folder and see summary information on their ROs. In future versions, we plan to find out with the users what else would be useful to have in the Dashboard.
- A Getting Started wizard-like page helps the user go through this process (as depicted in Figure 4).

Figure 4: First Prototype Architecture

Working with Dropbox folders and the ROSRS. Users register a single folder in their Dropbox as being their ROs store or container, i.e.: the folder in which all their ROs are contained in. Within this folder, each folder is considered to be a single RO.

For example, in Figure 5, AstroROs is the folder specified by the user as being their ROs folder. Within this there are three ROs that will be synced with the ROSRS - KpDrop, KpMap and Kptest.

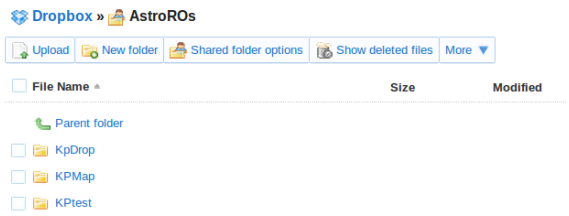

Figure 5: First Prototype Architecture

As part of the application development, we made the following design decisions:

Handling synchronisation. The Dropbox API does not provide adequate capabilities for synchronisation use cases[8]. Hence, we will have to poll the Dropbox API (but not too frequently!) and check for changes. To aid in this, the metadata about a directory contains a hash field that changes whenever the contents of the directory change. The API can return back an HTTP 304 when the metadata hasn't changed. Nevertheless, this only tells you when the flat list of files within the directory has changed and does not take into account any nested directories.

Table 4 summarizes the mapping between RO concepts, Dropbox concepts, ROBox objects and ROSRS concepts.

Table 4: Mapping between between RO concepts, Dropbox concepts, ROBox objects and ROSRS concepts

RO Concept	in Dropbox	in ROBox	in ROSRS	Notes
ROs Location	A folder that contains multiple ROs	DropboxResearchObjectContainer object / "ROs Location" in UI	A Workspace	–
RO	A folder within the ROs location folder	ResearchObject object / "Research Object" in the UI	A Research Object	Note the difference between the ROs location folder and a single RO folder within this
RO Manifest	*manifest.rdf* within RO folder	Cached as part of a ResearchObject object - shown in a structured manner in the UI	*manifest.rdf*	Assumed to always be at the root of an RO
RO Resources	Files and folders within an RO folder	DropboxEntry objects	Resource files	Mimics the exact same layout and structure

Technical implementation status. ROBox is built using the Ruby on Rails 3 framework[9]. Bundler[10] is used to manage and load Gem[11] dependencies. Phusion Passenger[12] (also known as mod_rails) can be used to deploy the application on an Apache based web server.

Development and deployment instructions. See Appendix B.

2.2.5 Tests

We performed tests on the prototype at different levels.

REST API tests

First, at the level of ROSRS, the following simple tests have been conducted:

- Create workspace
  - Create new RO
    - Create new RO version
      - Add two text files
      - Get list of all ROs
      - Get list of all versions of an RO
      - Get zipped content of a version of RO
      - Get manifest of a version of RO
      - Get files metadata
      - Get files content

[8] See for example this forum thread: `http://forums.dropbox.com/topic.php?id=34823`
[9] `http://rubyonrails.org/`
[10] `http://gembundler.com/`
[11] `http://rubygems.org/`
[12] `http://www.modrails.com/`

            - Get directory content list
            - Get zipped content of a directory
            - Update files
            - Update manifest
            - Update malformed manifest
            - Create a new RO version as a copy of an existing one
            - Delete files
        * Delete RO version
    - Delete RO

- Delete workspace

End-to-end functional tests

The end-to-end test suite was designed to perform automated testing of the proposed DropBox-based access to the back-end dLibra workflow respository, via the ROBox interface. As such, it uses shell scripts to perform the kinds of operation that a user might perform: copying files, examining files, deleting files, etc. A Research Object is presented as a directory tree, the root of which contains a manifest file named "manifest.rdf", which is populated with metadata about the RO.

For this iteration, operations submitted via file system updates would be reflected back to the user via updated metadata in the manifest.rdf file. In order to detect and test these metadata updates, a set of command line tools for querying RDF data, part of the OpenJena RDF toolkit (`http://openjena.org/`), have been used. A complete copy of the required tools has been lodged in the prototype test suite repository (`https://github.com/wf4ever/prototype1-testsuite`), so there are no external dependencies for running the test suite (other than installed Java).

To run the test suite, the repository should be cloned to a test machine with a suitably configured DropBox account (details in the project's private wiki area), and the file "TestConfig.sh" should be edited so the line

`DropBoxDir="..."`

reflects the location of the local dropbox folder where ROs will be created.

Each test case is implemented as a separate script (e.g., TestCreateRO.sh), and a master script "TestAll.sh" will be created to run all of the test cases. Each test script returns an exit code of zero if the test (or all of the tests) pass, or non-zero if there is a failure. This is intended to allow the tests to be integrated into a continuous integration environment.

The original test plan (below) called for a number of tests that would exercise various aspects of creating and manipulating ROs via a DropBox-synchronized file system. In practice, constraints on the first iteration ROBox implementation has meant that the metadata synchronization would work in only one direction, meaning that some of the proposed tests would not be implementable. The test plan was accordingly reined back to cover just the following tests:

- create research object
- add data to research object
- modify data in research object
- delete data in research object

The remaining tests will be fed forward into the next iteration as a contribution to the ongoing discussion about how the interface should operate.

A secondary problem has been that the DropBox API makes it hard to automatically detect changes in a timely fashion. As a consequence, the test suite has to contain long delays, which makes it difficult to use

as an effective test suite. At the time of writing this, we are working on a mechanism we can use to speed up the testing without over-stressing the DropBox API. In the longer term, we'll need to find a better way of notifying the ROBox interface that there are updates to be processed.

At the time of writing this, only the "create research object" test has been implemented, and has not yet been run successfully due to the DropBox update notification problem. We now have worked out and trialled a fix for the update notification problem, but this still needs to be fully deployed and tested in the test suite. Once the initial test is working, the others will be very quick to implement: we anticipate that the planned test suite will be completed and working by the beginning of June 2011.

2.3 Relation to Architecture and Reference Implementation

The methodology for the Wf4Ever architecture design is a co-evolutionary approach, where design is coupled with requirements gathering. The principles of software design for scientists have been established in previous work [RG09] and in the field of participatory design. The challenges are [EBDN03]:

1. Is it possible to more directly couple design of infrastructure features to the design of application features?
2. How can this more direct coupling exist when the applications the infrastructure supports don't yet exist or cannot be built without the infrastructure itself?
3. Could the context of either users or the use of these unknown applications have an important impact on the features we choose?
4. How can we avoid building a bloated system incorporating every conceivable feature, while ensuring we have a system that will not be constantly updated (and so repeatedly broken) throughout its lifespan?

Fortunately in Wf4Ever we are working from existing systems with established user communities, notably dLibra, myExperiment and Taverna. Through the development of the sandbox we are able to work closely with our users, combining requirements gathering with architecture design. A the same time, we have clear indications of architectural style and this provides a framework for this conversation. In particular, the focus in ROBox around the API design is a key component of our approach.

Hence, on the one hand, the Sandbox offers the infrastructure to test instantiations of the Wf4Ever architecture. On the other hand, the continuous testing of the software deployed from users and developers will inform the architecture specification for refinements and improvements. Moreover, the Sandbox is the place where the final reference implementation will be deployed.

3 Sandbox Walkthrough

This section provides information to help users get started with using the Sandbox. The goal is to familiarize users with the two ways of interacting and using the Sandbox and to enable them to better understand how to use the software deployed.

3.1 Live Demo Site

The Sandbox hosts all the services described in Section 2. These services are available to developers and users for testing and using them. The server is administered by PSNC and can be accessed by developers through SSH. Currently, there are 2 accounts: *dlibra* for dLibra and ROSRS development and *robox* for ROBOX development.

The users can interact with the Sandbox using its web interface. Figure 6 depicts the default Sandbox page `http://sandbox.wf4ever-project.org/`, which contains a brief description of what is installed on the server and how it can be used.

In particular, users can use the Sandbox to perform the following tasks:

- Connect their DropBox account to dLibra, using ROBox. This is the basic method of using the software. To do so, users must open the ROBox initial page `http://sandbox.wf4ever-project.org/robox` and follow the instructions that appear. A general description of how to use ROBox can be found in 2.2.4.
- Test the ROSRS REST interface. The interface is available under the address `http://sandbox.wf4ever-project.org/rosrs2`. Sample code that tests the interface can be found in ROSRS source hosted on Github[13].
- View the Research Objects as they are stored in dLibra and manipulate them. Users can run the dLibra Editor's Application available under the address `http://sandbox.wf4ever-project.org/dlibra-editor/editor.jnlp`. To ensure data privacy, login credentials are sent to users on request.
- Download Sandbox disk image. Sandbox disk images are available on the Sandbox itself for download for those users that need more interaction options or want to configure the software according to their needs. For more information, see paragraph below.

3.2 Sandbox Disk Image

The disk image of Sandbox is available for download, so that users can create their own virtual machines with the same configuration as Sandbox. In this way, users can test and use the latest version of Wf4Ever software without the need to download and configure each piece of software separately. In particular, we envision two main usages of the VM image: for developers, they can benefit with such pre-configured environment that will allow them to make local tests, including making changes in the software configuration as necessary, and without relying in the connection with external servers. For end-users, they can benefit from the image specially when they need to create and use huge amounts of data (e.g., input data for workflows in ROs), either for testing or for production, and when they need to rely on the preservation of that data.

The disk images are available in VMDK[14] format, supported by VMware[15], VirtualBox[16] and QEMU[17], among

[13] `https://github.com/wf4ever/prototype1-dlibra/blob/master/src/test/ruby/test2.rb`
[14] Virtual Machine Disk Format, `http://www.vmware.com/technical-resources/interfaces/vmdk.html`
[15] `http://www.vmware.com/`
[16] `http://www.virtualbox.org/`
[17] `http://www.qemu.org/`

Figure 6: Sandbox welcome page

others. To download them, users should go to `http://sandbox.wf4ever-project.org/images.html`.

The disk images are created every time the software installed on the Sandbox is changed significantly. They are created as copies of the Sandbox disks, with the following modifications:

- System user passwords are reset.
- Database user passwords are reset.
- Apache Tomcat user passwords are reset.
- dLibra content is reset to sample data.
- Disk images are deleted.

Users can log in to the system and to the MySQL database using the credentials listed in table 5. Please note that after running the downloaded Sandbox disk image for the first time, users should run the configuration script in `/home/root/first-time-configuration.sh`.

Username	Password	Comment
root	root123	System root
dlibra	dlibra123	System user for dLibra and ROSRS configuration
robox	robox123	System user for ROBox configuration
root	mysql123	MySQL root user

Table 5: User credentials for Sandbox Disk Image

4 Sandbox Maintenance

The Sandbox will be continuously evolving: different services and software components will be developed throughout the project lifetime, which will need to be integrated; additional software will be found relevant for the development of Wf4Ever reference implementation; new requirements for the running environment will be derived from the new software, etc.

Hence, the description provided in this document should be considered as a snapshot of the Sandbox configuration that corresponds to the result of the first iteration of Task 1.2, concerning the deployment of the Sandbox and continuous tracking of technologies relevant for scientific workflows preservation.

This section elaborates on the procedures envisioned for keeping the documentation about the Sandbox updated and accurate, as well as for upgrading the software deployed.

4.1 Documentation

The documentation of the Sandbox will be maintained up to date in the project wiki. As host of the Sandbox and responsible for its deployment, PSNC will be in charge of updating the core documentation as necessary. That is, changes in the running environment, the generic information of software deployed, and the walkthrough guides for using the live demo site and the VM image.

Details of the services and software components developed within the project, will also be maintained in the project wiki. However, keeping them up to date will be responsibility of the partner in charge of the development.

4.2 Upgrade Procedure

The software developed during the project will be deployed/updated in the Sandbox after each internal release of the software. In most cases the update will be performed by the partner responsible for the particular software module in a way specific for the module (for example via SSH or by uploading .war archive to preconfigured Tomcat instance). The system software (operating system, database etc.) will be updated by the PSNC administrator as needed.

The aim is to have the latest stable version of the prototype software running on the Sandbox to allow interested parties to interact with it. After update of the software on Sandbox a new VM image of the Sandbox will be prepared by PSNC and published for download. Each time the new Sandbox image will be published it will contain only some example data. At this stage it is not planned to provide upgrade instructions for data migration between VM images of the Sandbox. Such instructions may be provided in the future when the software running in the Sandbox will be in a mature stage. Therefore the Sandbox image should be used for local interaction with the prototypes or for development of applications and not for real research data storage.

Anyone using the live Sandbox instance must take into account the following:

- Data uploaded to live Sandbox instance may be deleted at any time as a result of the software upgrade. The live Sandbox should not be used as the only place to store important data.
- The safety and security of the data uploaded to live Sandbox cannot be guaranteed. The Sandbox will be based in prototype software which may potentially contain bugs leading to unauthorized access to data or to data modification/corruption.

5 Conclusions

The Sandbox provides a test environment where users and developers can try and interact with software and technologies relevant for Wf4Ever. It includes, both services and software components (e.g., prototypes) that are being developed within the project, as well as selected software relevant to the development of components or applications based on scientific workflows, their sharing, within social networks, semantic technologies, and digital libraries. Moreover, the Sandbox provides the infrastructure where the Wf4Ever reference implementation will be deployed.

In this document we have presented the results of the first iteration on the deployment of the Sandbox. We have provided a description of the Sandbox configuration, implemented as a virtual machine, and the software deployed. In particular, we have focussed on the first prototype developed within the project, which implements basic services supporting the interaction of scientists with Research Objects. It enables the connection of a Dropbox account to a Wf4Ever RO service. The idea is to support an automated publication and synchronization of ROs between a DropBox folder and the dLibra system.

We have also provided a walkthrough the Sandbox. There are two ways of using the Sandbox: first through a live demo site and second by downloading and installing locally a VM image. The first mode allows users to have a have a quick access to the software deployed, and use some exemplary data to interact with ROs and other concepts being specified as part of the project. The second mode enables a more deep testing of the software, create and use larger amounts of data, keeping control of data and the performance of the machine. For both cases, we provided the necessary information to help users get started with using the Sandbox.

Finally, as the Sandbox will be continuously evolving throughout the project lifetime, we presented the procedures that we will perform in order to keep an up to date documentation and to upgrade the software deployed, e.g., when new versions become available, when new components are being developed, or when new selected technologies will be tested/integrated.

Appendices

A ROSRS REST Interface

General concept

This chapter describes the REST interface of Research Object Store and Retrieve Service (ROSRS). The service enables manipulation of Research Objects stored in dLibra digital library.

Usage of HTTP Status Codes

For each request to ROSRS REST interface a HTTP status code will be returned. Below is a list with all supported status codes:

200 Ok - GET or POST request has been fulfilled

201 Created - POST request has been fulfilled and resulted in a new resource being created

204 No Content - DELETE request has been fulfilled

400 Bad Request - malformed request

401 Unauthorized - user authentication missing

403 Forbidden - user does not have access rights to requested request

404 Not Found - request not supported by ROSRS

409 Conflict - trying to create resource that already exists

500 Internal Server Error - an error occurred in ROSRS or dLibra

Please note:

- Error codes 401 (Unauthorized), 403(Forbidden), 404 (Not Found) and 500 (Internal Server Error) can be returned as a response to any request described below. As their meaning is not specific to any service, they are omitted in the interface description.

Authentication

Each request to ROSRS must contain user credentials, i.e. username and password. Workspaces can be created and deleted only by users with administrator rights. All other requests must use workspace id as a username together with the password that was provided when creating the workspace.

Services provided

Create new workspace

Description	Creates new workspace with the specified workspace id.
URL	`http://.../workspaces`
HTTP Method	`POST`
Content	(text/plain) workspace id in first line and password in second.
Parameters	-
Returns	201 (Created) when the workspace is successfully created.
Errors	409 (Conflict) if the workspace id is already used.

Delete workspace

Description	Deletes the workspace.
URL	`http://.../workspaces/WORKSPACE_ID`
HTTP Method	`DELETE`
Content	-
Parameters	WORKSPACE_ID - id of workspace to delete.
Returns	204 (No Content) when the workspace is successfully deleted.
Errors	-

List Research Objects

Description	Returns list of links to research objects. Output format is TBD.
URL	`http://.../workspaces/WORKSPACE_ID/ROs`
HTTP Method	`GET`
Content	-
Parameters	WORKSPACE_ID - workspace id.
Returns	200 (OK) containing a list of research objects.
Errors	-

Create Research Object

Description	Creates new Research Object with the given identifier.
URL	`http://.../workspaces/WORKSPACE_ID/ROs`
HTTP Method	`POST`
Content	(text/plain) RO id.
Parameters	WORKSPACE_ID - workspace id.
Returns	201 (Created) when the RO is created.
Errors	409 (Conflict) if the RO id is already used.

List versions of a Research Object

Description	Returns list of versions of this research object.
URL	`http://.../workspaces/WORKSPACE_ID/ROs/RO_ID`
HTTP Method	`GET`
Content	-
Parameters	WORKSPACE_ID - workspace id, RO_ID - RO id.
Returns	200 (OK) with an rdf file in response body containing OAI-ORE aggregates tags.
Errors	-

Create new version of a Research Object

Description	Creates new version of a Research Object. If URI of another version is provided, the new version is created as a copy of the other one.
URL	`http://.../workspaces/WORKSPACE_ID/ROs/RO_ID`
HTTP Method	`POST`
Content	(text/plain) Version id in first line and base version URI in second (optionally).
Parameters	WORKSPACE_ID - workspace id, RO_ID - RO id.
Returns	201 (Created) when the new version is created.
Errors	409 (Conflict) if the version id is already used.

Delete a Research Object

Description	Deletes the Research Object.
URL	`http://.../workspaces/WORKSPACE_ID/ROs/RO_ID`
HTTP Method	`DELETE`
Content	-
Parameters	WORKSPACE_ID - workspace id, RO_ID - id of RO to delete.
Returns	201 (Created) when the new version is created.
Errors	204 (No Content) when the RO is successfully deleted.

Get Research Object metadata

Description	Returns metadata of RO version as manifest.rdf file.
URL	`http://.../workspaces/WORKSPACE_ID/ROs/RO_ID/RO_VERSION_ID`
HTTP Method	`GET`
Content	-
Parameters	WORKSPACE_ID - workspace id, RO_ID - RO id, RO_VERSION_ID - version id.
Returns	200 (OK) with manifest.rdf file in response body.
Errors	-

Get Research Object content

Description	Returns zip archive with contents of RO version.
URL	`http://.../workspaces/WORKSPACE_ID/ROs/RO_ID/RO_VERSION_ID ?content=true`
HTTP Method	`GET`
Content	-
Parameters	WORKSPACE_ID - workspace id, RO_ID - RO id, RO_VERSION_ID - version id.
Returns	200 (OK) with zip file in response body.
Errors	-

Update metadata of a Research Object

Description	Updates metadata of a version of RO (manifest.rdf file).
URL	`http://.../workspaces/WORKSPACE_ID/ROs/RO_ID/RO_VERSION_ID`
HTTP Method	`POST`
Content	(application/xml+rdf) Manifest.rdf.
Parameters	WORKSPACE_ID - workspace id, RO_ID - RO id, RO_VERSION_ID - version id.
Returns	200 (OK) response code if the descriptive metadata was successfully updated.
Errors	400 (Bad Request) if manifest.rdf is not well-formed, 409 (Conflict) if manifest.rdf contains incorrect data (for example, one of required tags is missing).

Delete a version of a Research Object

Description	Deletes a version of a Research Object.
URL	`http://.../workspaces/WORKSPACE_ID/ROs/RO_ID`
HTTP Method	`DELETE`
Content	-
Parameters	WORKSPACE_ID - workspace id, RO_ID - RO id, RO_VERSION_ID - id of a version to delete.
Returns	204 (No Content) when the version of RO is successfully deleted.
Errors	-

Get a file or directory metadata

Description	Returns requested file metadata. If requested URI leads to a directory, returns rdf file with list of files in this directory.
URL	`http://.../workspaces/WORKSPACE_ID/ROs/RO_ID/RO_VERSION_ID /path/to/file`
HTTP Method	`GET`
Content	-
Parameters	WORKSPACE_ID - workspace id, RO_ID - RO id, RO_VERSION_ID - version id.
Returns	200 (OK) with an rdf file in response body.
Errors	-

Get a file or directory content

Description	Returns file content. If requested URI leads to a directory, returns zip archive with contents of directory.
URL	`http://.../workspaces/WORKSPACE_ID/ROs/RO_ID/RO_VERSION_ID /path/to/file?content=true`
HTTP Method	`GET`
Content	-
Parameters	WORKSPACE_ID - workspace id, RO_ID - RO id, RO_VERSION_ID - version id.
Returns	200 (OK) with a content file in response body.
Errors	-

Add or update a file

Description	Used for adding and updating files, also results in the manifest.rdf modification (the part with the structural metadata).
URL	`http://.../workspaces/WORKSPACE_ID/ROs/RO_ID/RO_VERSION_ID /path/to/file`
HTTP Method	`POST`
Content	New or updated file.
Parameters	WORKSPACE_ID - workspace id, RO_ID - RO id, RO_VERSION_ID - version id.
Returns	200 (OK) response code if the file was successfully created/updated.
Errors	400 (Bad Request) if request has empty body - this means that empty directories are not supported.

Delete a file

Description	Used for removing files from research object, also results in the manifest.rdf modification (the part with the structural metadata).
URL	`http://.../workspaces/WORKSPACE_ID/ROs/RO_ID/RO_VERSION_ID /path/to/file`
HTTP Method	`DELETE`
Content	-
Parameters	WORKSPACE_ID - workspace id, RO_ID - RO id, RO_VERSION_ID - id of a version to delete.
Returns	204 (No Content) when the file is successfully deleted.
Errors	-

B Development and Deployment Instructions

B.1 ROSRS

The source code is available at `https://github.com/wf4ever/prototype1-dlibra`. dLibra server location and directory used for storing workspaces should be configured in *src\main\resources\connection.properties* file.

A dLibra instance used for demonstration purposes is available at host `sandbox.wf4ever-project.org` (port number 10051 and directory 3, as originally configured).

Once that has been configured the project can be built (mvn package) and deployed (rosrs2 servlet).

B.2 ROBox

Installation (for Production use)

Dependencies

- Ruby 1.9.2 and development headers (ruby-dev)
- MySQL 5 and development headers (mysql-dev)
- Git 1.7+

Set up

- `gem install bundler`
- `git clone git://github.com/wf4ever/prototype1-dropbox.git`
- `bundle --deployment`
- Set up config/database.yml
- Set up config/settings/custom.yml
- `rake db:setup && rake db:migrate && rake db:seed`
- `jammit`
- `gem install passenger`

- Set up passenger in Apache
- Set up application in Apache

Running

Test the web server

```
rails server RAILS_ENV=production
```

Then open http://localhost:3000 in your browser.

How to run the background sync jobs First, run the background job worker:

```
ruby script/delayed_job start
```

This will run in the background until you stop it using:

```
ruby script/delayed_job stop
```

To then submit a fresh batch of sync jobs:

```
rake robox:sync_jobs
```

Development

Follow the installation instructions above, except:

- Do `git clone git@github.com:wf4ever/prototype1-dropbox.git` instead.
- Run `bundle install` instead of `bundle --deployment`.
- Don't do the Passenger and Apache set up.

To run the tests

```
rake spec
```

To have tests running continually in the background

```
autotest
```

This will run the appropriate test(s) when file(s) are changed.

To run a local development server

```
rails server
```

Then open http://localhost:3000 in your browser.

To completely delete your database and create a fresh new one

```
rake db:drop && rake db:create && rake db:migrate
&& rake db:seed && rake db:test:prepare
```

To re-annotate Models with database schema info

```
annotate --position=before --show-migration --show-indexes
```

To re-annotate the Routes file with a list of routes

```
annotate --routes
```

To generate the diagrams in the doc folder

```
rake diagram:all
```

Please note that for the above procedures to work, `dot`, `neato` and `sed` must be available on the command line.

References

[EBDN03] W. Keith Edwards, Victoria Bellotti, Anind K. Dey, and Mark W. Newman. The challenges of user-centered design and evaluation for infrastructure. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '03, pages 297–304, New York, NY, USA, 2003. ACM.

[RG09] David De Roure and Carole Goble. Software design for empowering scientists. *IEEE Software*, 26(1):88–95, January 2009.