Wf4Ever: Advanced Workflow Preservation Technologies for Enhanced Science

STREP FP7-ICT-2007-6 270192

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

D1.1 Setup of software development technologies

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

Deliverable Co-ordinating Institution: **PSNC**

Other Authors: **Adam Dudczak (PSNC)**

This deliverable describes the connection details of the SVN repository for storage and version management of the code developed within the project, the continuous build infrastructure, the proposal for build specifications, automated tests, and software releases.

Document Identifier:	Wf4Ever/2010/D1.1/v1.0	Date due:	December 31, 2010
Class Deliverable:	Wf4Ever FP7-ICT-2007-6 270192	Submission date:	December 31, 2010
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	13-12-2010	Raul Palma	Created Document
0.2	14-12-2010	Raul Palma	Introduction
0.3	15-12-2010	Raul Palma	Software Development Infrastructure
0.4	15-12-2010	Marcin Werla	Deployed Software Development Infrastructure
0.5	16-12-2010	Raul Palma	Update Software Development Infrastructure
0.6	17-12-2010	Raul Palma	Merged inventories of software development tools and software components
0.7	21-12-2010	Adam Dudczak	Updated Software Development Infrastructure
0.8	23-12-2010	Adam Dudczak	Updated Software Development Guidelines
0.9	28-12-2010	Raul Palma	Conclusions, updates and revision of whole document
1.0	30-12-2010	Raul Palma	Update according to QA comments and included anonymous access description

Executive Summary

In this deliverable, we describe the configuration of the technologies that have been selected and installed to support the tasks of distributed and collaborative software development in Wf4Ever. The development of software will take place both in the context of the technical workpackages (WP2-WP4) and in the context of the application workpackages (WP5-WP6).

The deliverable also describes the proposal for build specifications and testing, as well as principles of practice for software development to be followed in Wf4Ever. Finally, this deliverable includes two inventories:

- The inventory of development tools that each partner uses to develop software internally
- The inventory of components from each partner, which can contribute to the implementation of the reference implementation of Wf4Ever

It is important to note that the software development infrastructure and guidelines that we present in this document are very likely to evolve as new requirements emerge during the project lifetime. Hence, we will maintain this information alive and updated in the project wiki.

Contents

List of Tables

1 Introduction

One of the main outcomes of Wf4Ever is a reference implementation enabling the preservation and efficient retrieval of scientific workflows across a range of domains. Such implementation will extend one of the most widely deployed scientific workflow sharing infrastructures (myExperiment) with preservation capabilities that consider the complexity of scientific workflows and their related objects. To this end, the system will rest on digital libraries for scientific workflows, and novel software components will be developed for (i) workflow decay analysis, abstraction and comparison (WP2); (ii) Research Object evolution, personalised recommendations and collaboration between scientists (WP3); and (iii) integrity and authenticity management based on provenance models of workflow-related Research Objects (WP4).

The development of the reference implementation will be a continuous process of integration of the foundational systems that provide support to scientific workflow sharing and digital preservation, together with the basic components and services developed in the technical workpackages (WP2-WP4). Furthermore, the team of developers that will participate in the development of this reference implementation reside in separate locations in the world, spanning different time zones. Therefore, in W4Ever we plan to develop prototypes using an agile software development methodology like SCRUM. This approach builds on iteratively and incrementally developing software through the continuous evolution of requirements and solutions and on stimulating the collaboration between self-organizing, cross-functional teams.

In this deliverable we describe the initial software development infrastructure that we have deployed in order to support the development tasks in Wf4Ever. For the selection of this infrastructure, we considered its suitability to an agile software development methodology as well as its suitability to support the collaboration of a distributed team of developers. Moreover, we also considered the software development tools used by each partner as well as the underlying technologies of the software components from each partner that may be integrated in the reference implementation. For this task, we created an inventory of development tools and software components, where each partner provided their input.

Additionally, in this deliverable we describe the proposed build specifications, continuous integration mechanisms and automated tests, support for software releases and principles of practice for software development to be followed in Wf4Ever.

The software infrastructure and the guidelines that we describe in this document are very likely to evolve as new requirements emerge during the project lifetime. Hence, an alive and updated description of the current state of the deployed infrastructure and used guidelines will be maintained in the project wiki.

The remainder of this document is organized as follows: Section 2 presents the software development infrastructure. Next, in Section 3 we present the proposed guidelines and principles, and we conclude in Section 4. Additionally, in the Appendix, we include the inventories of software development tools and software components that we collected from all partners.

2 Software Development Infrastructure

This section details the tools and mechanisms that are being provided to developers within Wf4Ever to help them produce the necessary software artifacts, ensuring adequate communication, collaborative writing of source code, and means to integrate and test the individual components, subsystems and the system as a whole.

2.1 Source Code Repository

Software development in Wf4ever involves many software components that need to be orchestrated together to build a reference implementation. Moreover, development tasks will be carried out by a distributed team involving people from different institutions and different countries. Keeping track of all the software pieces and people involved in such a collaborative scenario is a major effort, specially as the number of people involved grows. Hence, it becomes necessary to use some kind of source code management tool. Such kind of tools are, nowadays, a core component of most software development projects.

There are several tool options, commercial and open source, offering similar functionality. We have chosen at this moment the open source tool Subversion as the source code management tool, due to its maturity (it is the current open source repository of choice), cheap branching abilities, and the fact that it can be used through HTTP or HTTPS. However, we are open to use a distributed system, such as Git[1], in the near future if the project needs and developers preferences will suit better. For example, according to Tables 1 and 2, Wf4Ever partners currently use different tools. Some partners use different tools in different projects (relevant for Wf4Ever) or even in the same project, including SVN (PSNC), Git and Google code[2] (myGrid project[3] from OXF, UNIMAN) and RubyForge[4] (myExperiment[5] and Biocatalogue[6] projects from OXF and UNIMAN).

2.1.1 Wf4Ever Repository

- The **Subversion (SVN)** instance used in the project as a repository is hosted by PSNC and is available at `https://svn.man.poznan.pl/svnroot/wf4ever/`. The version of SVN is **1.5.1**.
- The access to the SVN instance is described in Section 2.7 of this document.
- One of the information sources about SVN can be the ebook titled "Version Control with Subversion" that is freely available at `http://svnbook.red-bean.com/` (please use ebook for Subversion ver. 1.5).
- Exemplary comparison of different SVN clients can be found at `http://en.wikipedia.org/wiki/Comparison_of_Subversion_clients`.

2.1.2 Web Access to the Wf4Ever Repository

- In addition to the HTTPS access to the repository, there is also a dedicated tool set-up which should facilitate browsing through the repository, monitoring of changes and some basic repository analysis. This tool is named **Fisheye** and its documentation can be found at `http://confluence.atlassian.com/display/FISHEYE/FishEye+User%27s+Guide`.

[1] `http://git-scm.com/`
[2] `http://code.google.com/`
[3] `http://www.mygrid.org.uk/`
[4] `http://rubyforge.org/`
[5] `http://www.myexperiment.org/`
[6] `http://www.biocatalogue.org/`

- Fisheye is integrated with another tool named **Crucible**, designed to support remote code reviews (see Section 2.2.2). Crucible documentation is available at `http://confluence.atlassian.com/display/CRUCIBLE/Crucible+User%27s+Guide`.
- Wf4Ever instance of those services is available at `https://fisheye.man.poznan.pl/`.
- The access to those services is described in Section 2.7 of this document.

2.2 Issue Management

The purpose of Issue Management is to ensure that any concerns recognized during a project are addressed in a timely manner and do not go unresolved until they become major problems[7].

Software development projects typically use an issue management system in order to improve the code quality and speed of development but can be also used as a communication tool.

In Wf4Ever, such a system would be particularly useful for the development of the reference implementation that will involve the integration of many different components and the collaboration of many different developers.

Ideally, issues in such a management system will be linked to the source code generated to resolve the issue, the release containing the fix, discussions and any other artifacts associated with the issue.

We have chosen JIRA [8] as the issue management system to be used in Wf4Ever. JIRA is a very mature, robust but still flexible issue tracker. It supports Agile development using methodologies like Scrum and XP. JIRA is widely used by many Open Source projects, e.g. most of the projects developed by Apache Software Foundation are using JIRA. Additionally, JIRA is also used by some Wf4Ever partners, e.g., PSNC, OXF and UNIMAN (myGrid project). Other tools used by partners include Fogbugz[9] and RubyForge (myExperiment and Biocatalogue from OXF and UNIMAN) and Trac[10] by UPM (see Tables 1 and 2). However, as JIRA has all necessary features, all partners agreed to use it for the project.

2.2.1 Wf4Ever Issue Management System

- Wf4Ever's **JIRA** instance is enriched with **GreenHopper**, a tool supporting agile project management.
- JIRA documentation can be found at `http://confluence.atlassian.com/display/JIRA/JIRA+User%27s+Guide`, and GreenHopper documentation can be found at `http://confluence.atlassian.com/display/GH/GreenHopper+User%27s+Guide`.
- There are also several plugins allowing to use JIRA directly from software IDE. Some of them are free. More information can be found at `http://www.atlassian.com/software/ideconnector/`
- Wf4Ever instance of JIRA is available at: `https://jira.man.poznan.pl/jira/browse/WFE`.
- The access to Wf4Ever's JIRA instance is described in Section 2.7 of this document.

2.2.2 Integrated Wf4Ever Software Development Infrastructure

Wf4Ever's JIRA and SVN instances are linked together with the Fisheye [11] + Crucible [12] tool. Crucible is a great tool which allows to perform efficient code reviews by local as well as by distributed teams. This means

[7] `http://www.pmhut.com/issues-management-process`
[8] `http://www.atlassian.com/software/jira/`
[9] `http://www.fogcreek.com/FogBugz/`
[10] `http://trac.edgewall.org/`
[11] `http://www.atlassian.com/software/fisheye/`
[12] `http://www.atlassian.com/software/crucible/`

that from the JIRA interface you can easily reach the source code modifications made while working on particular issues (tasks) defined in JIRA. On the other hand, while exploring the project repository in Fisheye, you can easily go to the details of issues (tasks) related to particular code changes. These tools can be used by all interested teams regardless of used programming language.

2.3 Build Process

The goal of the build process is to create a usable software from a set of source files. The process of building the software may take some time and have many associated tasks. To achieve high efficiency, this process must be automated with the help of build-tools like make, rake, Ant or Maven. Build automation is not only useful for developers (e.g., through command line or IDE) but also it is necessary to set up a continuous integration facility/server (this would be described in the following section). However, because tasks related to everyday developers activities are different from those implemented by continuous integration server it is necessary to have sufficient level of flexibility offered by build tool.

Build process may result in producing at least some of the following items:

- results of source code compilation (if applicable), packaged in the standard format related to given programming language (e.g., jar in Java);
- results of automated tests execution in form of an accessible report;
- other documents including release notes, various manuals and documentation of APIs created based on comments embedded in the source code (e.g. Javadoc).

Build tools may also allow for automated deployment of those artefact in various formats on a remote server.

We have chosen Maven 2.x [13] as the build tool for all Java-based Wf4ever components. Maven allows to work with Ruby code through support for JRuby but it might be more suitable to use native Ruby tools like Rake. Besides, Maven is already used by most of Wf4Ever partners, including PSNC, OXF and UNIMAN (myGrid project) (see Tables 1 and 2).

Maven implements the *convention over configuration*[14] paradigm to various aspects of software configuration management, e.g., standard directory layout, declarative project object model and predefined build lifecycles. Additionally, it also offers robust dependency management (including transitive dependencies), support for automated tests, static code analysis and documentation.

At the moment Maven is considered as an industry standard for building all kinds of applications starting from small Swing-based database front-end to large enterprise systems. Thanks to the availability of various plug-ins, Maven can be used also to build not only Java applications but also applications in different programming languages, such as C/C++. There are two great online books made available by creators of Maven:

- "Maven by Example" - `http://www.sonatype.com/books/mvnex-book/reference/public-book.html` and
- "Maven: The Complete Reference" - `http://www.sonatype.com/books/mvnref-book/reference/public-book.html`

Maven integrates well with most of the available continuous integration servers including Cruise Control, Hudson, Apache Continuum, Atlassian Bamboo, JetBrains TeamCity.

Finally, to achieve efficient dependency management, Maven introduces the concept of repositories which are used to store all necessary libraries [pro]. Hence, when the development of Wf4Ever reference implementation will advance, it may be necessary to create a dedicated Maven repository for project artifacts.

[13] `http://maven.apache.org`
[14] `http://en.wikipedia.org/wiki/Convention_over_configuration`

2.4 Continuous Integration

In Wf4Ever we will have a continuous integration environment for software development. According to Fowler [Fow06], Continuous Integration is a software development practice where members of a team integrate their work frequently (e.g., daily), so as to avoid that the repository mainline becomes too different from the developers' baselines. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

In order to implement an effective continuous integration, our infrastructure supports the following key practices proposed by Fowler:

- Maintain a single source repository
- Automate the build
- Make the build self-testing
- Everyone commits to the mainline as often as possible
- Every commit (to mainline) should be built
- Keep the Build Fast
- Test in a Clone of the Production Environment
- Make it easy to get the latest executables
- Everyone can see the results of the latest build
- Automate deployment

Although many of the items listed above are handled automatically by tools such as Maven combined with a source code repository, it also useful to use specific continuous integration tools for the management of the software development. Hence, in order to supplement Maven capabilities we have chosen the open source tool CruiseControl [15] (CC) for continuous integration in Wf4Ever. CC allows to broadcast results of the build process and to monitor repository in order to include changes made by developers as soon as they are committed. According to Tables 1 and 2, from the partners that use continuous integration facilities, CC is currently used by PSNC and Hudson CL[16] is used by myGrid project (OXF and UNIMAN). However, it was also agreed that CC has all necessary features.

2.5 Testing

Testing is about checking that the system built does what it is expected to do. In a continuous integration environment automated tests are typically included in the build process (as mentioned in section 2.3) to catch bugs more quickly and efficiently. In fact, the basis to create self-testing code, as proposed by Fowler [Fow06], is to have a suite of automated tests that can check a large part of the code base for bugs. The tests should be able to be launched using simple command and to be self-checking. Moreover, for a build to be self-testing the failure of a test should cause the build to fail.

Testing can be done on the various levels; however, there is no agreement among software engineers about exact scope (even on names) of test levels. In this document we will take rather practical approach and refer to three types of tests: unit, acceptance and performance.

Unit testing allows to verify whether all features which should be offered by a given class are in place. This effort also helps to validate whether the designed object model is usable and helps developers to define clear and understandable APIs.

[15] http://cruisecontrol.sourceforge.net/
[16] http://hudson-ci.org/

In terms of Java development **JUnit** [17] is one of the most widely used tools in this area. It is mature, allows to create test cases, group them into test suites and integrate them with various tools in order to implement more complex test cases.

In order to test some of the classes which require external resources it may be necessary to use mock objects. Mocks simulate behavior of real objects in order to test particular features of a given class. The two most widely used Java libraries in this domain are **EasyMock** [18] and **Mockito** [19]. Both of them offer similar functionality but it seems that it would be better to use the newer one which is Mockito.

The choice is much simpler when it comes to component developed in Ruby where built-in libraries should be used.

Acceptance (or functional) tests checks whether whole module works as it was defined in the specification. In general acceptance tests can be done by human tester but if possible this kind of tests should be also automated. This can be done using various tools; particular choice depends on the types of tested interfaces. Hence, at the moment no recommendation will be done here. However, if some parts of the reference implementation interface will be created using Ruby programming language it may be natural to use tools standard for this environment instead of using tools like Selenium [20] or HTMLUnit [21].

Non-functional or performance testing are used to verify whether the system (or its components) is able to work within given performance constraints. Depending on the type of developed application, non-functional requirement may be as important as the offered features. In terms of load generation and performance testing there are a lot of tools available in both Java and Ruby.

We have chosen **JMeter** [22] which is mature tool with a lot of plugins available and decent documentation. It allows to perform distributed stress testing and to exercise various user interfaces, e.g., WWW, Swing or SOAP.

2.6 Software Releases

Software release is the distribution of a software product, and it usually includes not only the executable program, but also software code, documentation, and support materials, e.g., release notes, configuration data.

The adoption of an agile software development methodology, as it is planned in Wf4Ever, usually increases the number of release events. Hence, in W4Ever we plan to have many internal or minor releases, and at least two major releases, one for each project development phase. Thanks to usage of tools like JIRA and Maven it is possible to keep cost of release management on very low level.

Detailed procedure of Wf4Ever software releases heavily depends on the architecture definition of the reference implementation, because this will determine which artifacts will be produced as an outcome of Wf4Ever development.

However software release procedure should be associated with general procedures that ensure that the developed product does what it is supposed to do. All found bugs should be fixed and changes related to them should be committed in the source code repository. The next step is to create tags and branches which will reflect the state of the code at the time of the release.

When the source code is ready, an agent responsible for releases should build the final project artifacts. This can be done manually or using continuous integration server. When creation of artifacts is finished, software should be deployed into a copy of the production environment. Information from issue tracker should be used to create the list of features implemented in a given release. A dedicated person or a group of people should verify the software quality using the list of implemented features and testing procedures provided by

[17] http://junit.org
[18] http://easymock.org/
[19] http://mockito.org/
[20] http://seleniumhq.org/
[21] http://htmlunit.sourceforge.net/
[22] http://jakarta.apache.org/jmeter/

developers in the description of the JIRA issue. If any bugs are identified, changes should be committed into source code repository and the whole procedure of building and verification of release should be repeated.

When previous steps are completed, all project artifacts should be placed in all interested locations including local mirrors of Maven repositories and, in case of public releases, the the project Website.

After successful deployment of the release, appropriate configuration files (e.g. POM files) should be updated in order to reflect that the team moves forward to work on the next release.

2.7 Access to Wf4Ever Software Development Infrastructure

General public

Access to the project SVN is open to the general public in read-only mode. Users should use anonymous as username and password. So the "read-authorized" URL is like this: `https://anonymous:anonymous@svn.man.poznan.pl/svnroot/wf4ever/`.

Additionally, access to the project JIRA instance is possible for anyone who will register here: `https://jira.man.poznan.pl/jira/secure/Signup!default.jspa`. Users are able to browse the JIRA contents and submit new bugs/issues.

Wf4Ever members and collaborators

In order to obtain full access to shared project software development infrastructure hosted at PSNC you have to do the following:

1. Go to `https://drawer.man.poznan.pl/hash/`.
2. On this page in the "INPUT" text field please enter a password which you would like to have for accessing the infrastructure.
3. Click CRYPT button.
4. A text in "OUTPUT" text field should appear. It is so called password hash, which allows you to send us your desired password with satisfactory security level.
5. Write an e-mail to Marcin Werla from PSNC (mwerla@man.poznan.pl) containing the following information about you:
   - name and surname
   - institution
   - e-mail
   - password hash copied from the OUTPUT text field (see previous step)

   IMPORTANT: Do not send the non-encrypted version of password! You will receive a confirmation e-mail with your username (most probably it will be <name>.<surname>) when the account will be ready.

3 Software Development Guidelines

Previous section briefly described the most important concepts related to development lifecycle and infrastructure. In this section, we will present the most important guidelines related to software development practice.

3.1 Using source code repository

While working with source code repository one should remember about few basic rules starting from what should be stored in SVN to project repository layout.

What to store
Everything that is necessary to do a build should be in the repository including not only code but also test scripts, properties files, database schema, install scripts, and third party libraries. That is, as stated by Fowler [Fow06], it should be possible to walk up to the project with a virgin machine, do a checkout, and be able to fully build the system.

What not to store
Repository must not contain any direct results of the build i.e. jar/war files, logs etc. In case of Maven driven projects adding svn:ignore to /target folder solves most of the problems.

Other recommendations
Most of recommendations presented here comes from [BMZ10], some of them were adjusted to tools presented in previous section.

- Only commit code that compiles. Code in the mainline of development (trunk) should pass continuous integration build process at any time.
- Comment each of your commits to SVN (at least with an issue identifier).
- To minimise conflicts (simultaneous modifications of the same part of source code), each module should be assigned to a single developer. If some conflicts will occur for example as a result of merging from another branch, conflicts should be solved by the developer who owns the conflicted module.
- Branch only when necessary. Before branching, consider the branch maintenance costs: builds, checks, updates against the trunk, periodic and final merges.
- Synchronise branches often. Regular merging of a branch with the main development is important, since it makes the final merge into the trunk much easier by keeping two codelines better aligned and reducing the number of potential conflicts.
- Merge as soon as the branch is complete. If possible, merge before the original codeline creates a new wave of changes.
- Merging should be done by the best prepared person. The owner of the target files and the person who made the changes are likely do a better merge than someone else.
- Do not establish independent copies of the code. Apart from check-outs, do not make editable or reference copies outside of your Version Control System (VCS), nor check in the copies of existing files as new ones. Use branches instead.

SVN Layout for individual modules

It is hard to determine the overall repository layout because this will depend on future architecture of the reference implementation. However each module should comply to the following best practices and SVN directory layout.

/trunk – It contains the main body of development.

- New modules should be started here.
- It is recommended that /trunk is always kept in a state in which it compiles and goes through continuous integration builds.
- In order to achieve consistency of /trunk, significant changes should be first completed within the branches and then merged into the /trunk

/branch – A branch is a special line of development of the source tree that starts with a specific copy of code.

- It is used to manage work that may be later merged back into the main trunk or originating branch. Using branches helps preserve the integrity of the trunk.
- Each branch is created as a separate folder within /branch. Unless the branch is by its nature limited to some part of the source, it is preferred that it represents a full snapshot of the whole /trunk (or originating branch). This provides an integral view of the branch, as well as a complete set of files to work with
- There are two basic scenarios for using branches:
  - Feature branches – used to implement particular feature which might not be ready before next release. Name of given branch should start with a word ”feature” and contain identifier of JIRA issue which describes given feature, e.g., **feature_WFE-1**.
  - Bugfix branches – each new release should result in creation of branch which contain a stable, editable snapshot of software at the time of release. This branch would be the base for further maintenance releases. Bugfix branches names should start with a prefix ”bugfix” followed by the particular version number associated with release e.g. **bugfix_1_0**.

/tags – Used for bookmarking or taking reference (non-editable) snapshots.

- Tags are used to mark specific releases or phases of development, in order to be able to retrieve and inspect a past release exactly as it was. Unlike branches, tags do not impose any maintenance costs, so they should be used liberally in order to track the release history.

3.2 Practical issue management guidelines

Issues related to new features [23] should be created by a person responsible for development of given module and then assigned to particular developers.

All problems encountered in the software should be reported as bugs in the issue tracker, no matter how small they seem. Bug reporter should comply to the following simple principles:

- Be precise;
- Be clear – explain the bug, so others can reproduce it. Provide all additional materials (i.e. database dumps) if it is necessary to reproduce given problem;
- One bug per report;
- No bug is too trivial to report – small bugs may hide bigger bugs.

[23] JIRA features predefined types of issues, among them you will find ”New Feature” and ”Bug”.

Bug report should be assigned to a person who is responsible for the module which most likely is causing the given problem. Teams can delegate to one person the responsibility of the bug assignment.
When an issue is solved, the developer should also resolve the given issue in the tracking system. In JIRA one of the steps during issue resolution includes the provision of the testing procedure.

- The given testing procedure or unit test should verify that the original bugs were fixed.
- Developers should provide a reference to unit test or document describing the testing scenario (this should be stored in the project source code repository). This information will be the base to verify whether implemented features works properly during the preparation of release.

Finally, testers (i.e, different users from the developer(s) who did the implementation of an issue) should work through the JIRA issues for the release marked as Resolved. These are tested to confirm they are solved, and if fixed marked as Closed, otherwise re-opened.

Using JIRA with FishEye

As it was mentioned in section 2.2.2, Wf4Ever's JIRA is integrated with other useful tools like FishEye and Crucible. The rules governing this link are straightforward:

- Before starting any code changes you must have a JIRA issue created, corresponding to the changes that you have to make.
- Each JIRA issue for the project will have a unique id with the following syntax: *WFE-<some integer>*. This issue id MUST be included in the message/comment that you provide while committing to the repository. For example it can look like this:
  WFE-123 - Final fix for the NullPointerException
  The issue id can be anywhere in the commit message, but it has to be separated with some characters, like space, from other message words.
- Such issue ids are automatically detected and utilised for linking between JIRA and Fisheye+Crucible.
- Because of the nature of this linking, it is not recommended to commit several different files/changes with one message containing several issue ids. So for example if you have three files named A, B and C, and you have worked on WFE-1 in file A, on WFE-2 in file B and on WFE-3 in file C, then instead of committing all three files at once with a general message like:
  WFE-1 WFE-2 WFE-3 - Now all this should work
  You should do three separate commits having more meaningful messages and only the proper issues ids. Of course if there will be some very closely related issues, then such separation may be not possible.

3.3 Working with Apache Maven

The following list of advices should help to avoid various problems which new Maven user may encounter.

- In order to avoid potential problems at the later stage of development it is recommended to install Maven in a path that does not contain any white spaces;
- It is recommended to include the following code analysis tools into the build lifecycle of all Java-based projects
  - Findbugs (`http://mojo.codehaus.org/findbugs-maven-plugin/`)
    - Detects common bugs in the code using static code analysis

  - PMD (http://maven.apache.org/plugins/maven-pmd-plugin/)
    - Also detects common errors in Java code
  - Cobertura (http://mojo.codehaus.org/cobertura-maven-plugin/)
    - Generates reports showing which parts of code are covered with automated tests
  - Checkstyle (http://maven.apache.org/plugins/maven-checkstyle-plugin/)
    - It makes easier to use the same coding style across the project

- Developers should remove potential errors detected by FindBugs and PMD
- Whenever possible you should use general conventions described in Maven documentations, e.g., project folder structure
- When a particular maven plugin is used, you should specify its version number to avoid problems with broken builds resulted from errors introduced in newer versions of plugins;
- Each partner should setup its own Maven dependency repository, to make all necessary artifacts available through the local network – this should speed up build process. For more information refer to [pro].
- In order to stop Maven from checking for plugin updates (it may also speed up build process), you may use ”–o” parameter. It forces Maven to work offline.

These and other recommendations can be found in the Maven books which were referenced in section 2.3.

3.4 Release management practices

In addition to the general overview presented in Section 2.6 regarding release management, in Wf4Ever we consider the adoption of the following practices proposed in the Development, release and testing procedures online document of MyGrid project[24]:

- Two weeks before a release preparation, a notification must be made through the Wf4Ever developer list about the planned code freeze. The actual release date would include additional time for testing, preparation and any last-minute changes, at least a week later.
- After these 2 weeks, the code freeze is announced.
- The maintainer (i.e., the person responsible for routine maintenance and processing a release) checks out the entire source tree from the relevant build project.
- The source tree is branched according to the release version:
  - The maintainer, working on the new branch, updates the dependencies by removing the snapshot versions (typically using find and sed).
  - Modules which are not updated for this release, are instead pulled in from their previously tagged/deployed versions.
  - The maintainer makes sure the code builds and runs locally.
  - Then the new poms, and other files such as plugin.xml files, are now committed. The maintainer makes a notes of the commit changes (using VCS logs, automated emails etc).
- A build is made for this particular version, checking out from the branch. This build forms the first internal release candidate which is distributed for testing.

[24] http://www.mygrid.org.uk/dev/wiki/display/developer/Development,+release+and+testing+procedures

  - At this point any bug-fixes or last minute tweaks are applied to the trunk, and merged across to the release branch.
  - After a release is prepared, and packaged, it is handed to the tester users.
- When a release is ready and sent out, the maintainer does a tag on the branch to mark the release .
- The build is modified to build from the release tag - this forms a golden master (i.e., a stable-versioned tagged version).
- The installers are created and uploaded from the golden master built.

3.5 Coding guidelines

Most of the guidelines presented in this section are extracted and summarized from [BMZ10], a very comprehensive description of various best practices in the domain of software development. Additionally, some of these guidelines are based on the Development, release and testing procedures online document of MyGrid project.

3.5.1 General rules

Use English – Results of whole work in Wf4Ever will be used and maintained in a multilingual context, because of that please use English in both source code (i.e. names of variables) and documentation

Keep your code simple – Please keep in mind the code is written once but read many times

Keep the implementation simple – "Smart" tricks, hacks or optimisations should always be avoided, unless there are specific functional requirements related to them

Avoid repetition – If given piece of the code is duplicated, you should encapsulate it and create new class/method/function

Some other complementary recommendations are:

- Write unit tests before coding, to verify bug and desired behaviour
- Add integration tests to the integration test module
- Ensure build is not broken before committing
- Monitor emails from CruiseControl about build and test failures - Fix build issues immediately
- If unsure about where to place code, contact the release manager to confirm whether your code needs a new module, and where it should go
- Never copy code. If code could be reused, feel free to do a refactoring to a common class/module. Coordinate with other team members
- External dependencies must be to stable versions pulled from an official Maven repository - and should not be added to the project as JARs or copied source code
- Update the @Author tag on any class modified

Also, to assure interoperability between various systems, all source code files should be encoded using only ASCII (Latin-1) characters [25]. Each class should be placed in its own separate file, named after the class. You should also avoid using white spaces in the names of all files.

[25] If necessary use `native2ascii` for Java classes.

3.5.2 Assuring code readability

Simplicity of the code is in fact quite complex issue, among other factors which influence code readability you should remember about the following things:

- Use meaningful identifiers using terminology that applies to the domain also helps a lot.
- Do not hesitate to use long names of variables if this adds clarity.
- Avoid abbreviations and acronyms.
- In documentation, describe the variable's purpose rather than its type.
- Avoid negated Boolean variable names e.g. isNotSet.

Code Conventions

Code conventions are programming guidelines that focus on the physical structure and appearance of a program, rather than its logic. They make the code easier to read, understand, and maintain [BMZ10].
We would like to suggest using well established standards in terms of code conventions:

- for Java – Sun's Java Code Conventions (JCC) [Mic99]
- for Ruby – as described in "The Unofficial Ruby Usage Guide" [Mac]

Below you will find set of suggestions mostly related to Java development, some of them may also be applied to Ruby.

- Imported classes should always be listed explicitly. Prohibit "*" in import statements
  - This gives you more control at the expense of more import statements, and IDEs easily organise imports
- In Java class variables should never be declared public. Use getter/setter convention if you have to expose content of given variable
- Avoid complex conditional expressions. Presence of complex *if* statements is a clear sign that code needs to be redesigned
- Avoid using magic numbers in the code. Instead declare numbers as named constants
- Logical units within a block of code should be separated by one blank line
- Use left-hand comparison style with constants or expressions
  e.g. `if ("example".equals(str))`, if str is null, the expression will return false instead of raising a null-pointer exception.
- Use properly configured code formatters before committing code into source code repository. This would help to avoid conflicts.
- Use tools like Findbugs and PMD often, this should help to keep your code clean.

Class and Interface declarations should be organised in the following order:

1. Class/Interface documentation.
2. Class or interface statement.

3. Class (static) variables in the order public, protected, package (no access modifier), private.
4. Instance variables in the order public, protected, package (no access modifier), private.
5. Constructors.
6. Methods (no specific order).

Comments in the source code

We suggest that all public classes and public and protected methods within public classes should be documented using the appropriate conventions and tools as described in [Mic] and [Mac].
According to [BMZ10] comments should be used to provide an overview of the code and additional information that is not readily available in the code itself.

- Comments should be in English
- Avoid any comments that are likely to become outdated as the code evolves
- If a comment no longer applies, you should modify or remove it
- Avoid replicating information that is obvious from the code
- Rather than commenting everything, consider if you could rewrite the code to make it clearer
- When using static variables you should describe why they are declared static
- Code that has been ”commented out” should be explained or removed
- All class files should contain formally required comments, as copyright header or author name
  - Comments in the beginning of the class should clearly state about role and responsibility of given class
- All classes, interfaces and, public and non-trivial internal methods should contain a descriptive documentation

3.5.3 Exception handling and logging

Most of the remarks presented in this section refers to Java development.

- Do not use null or similar value as an indicator of error. Instead of doing so, appropriate exception should be thrown.
- Avoid ambiguous use of try-catch blocks, catch specific exception (not general once like `java.lang.Exception`) and handle them appropriately.
- Catch block should not both log and throw another exception, because this doubles the log.
- Do not use `System.out` or `System.err` (especially in web applications), use appropriate logging facility, e.g., log4j or `java.util.logging`.

More examples on exception handling can be found here [McC06]. As was mentioned all caught exception should be properly handled and logged.
All available logging systems introduce several levels of logging. It is important to understand the meaning of each of those levels. The most important are described below.

FATAL – Serious errors that are likely to lead to premature application termination, e.g., lack of memory, lost database connection.

ERROR – Errors in the regular operation of the system which prevent normal program execution but are likely to allow the application to continue running.

WARN – Should be used to record transient problems or undesirable, unexpected or potentially harmful situations that are not necessarily errors.

INFO – Describes interesting runtime events, like startup, shutdown, receipt of user request. These messages should inform on the progress of the application at coarse-grained level during its normal operation.

DEBUG – These messages are produced by code added to debug an application for tracing its flow or internal data.

There are several Java libraries which offer logging features, including: Log4J [26], Apache Commons Logging [27] and `java.util.logging`. We recommend using Log4J as a logging facility in WF4Ever implementation as this is the most widely used and flexible library.

[BMZ10] contains several recommendations regarding logging practice:

- When logging, related data should be logged in a single message this help to understand what is happening during debugging.
- Logging system should be configured to inform not only about error message and stack trace. Log needs to contain information about date and time of the event.
- Log relevant and meaningful information only.
- Avoid logging at frequently or iteratively executed spots.
- Direct messages to different loggers according to purpose.
- Wrap time-consuming log messages in conditional statements
  e.g. `if (logger.isDebugEnabled())`

[26] `http://logging.apache.org/log4j/1.2/`
[27] `http://commons.apache.org/logging/`

4 Conclusions

Software development in Wf4Ever will be a very intensive activity throughout the whole project. The development of the reference implementation will involve the integration of existing software components that will be extended and innovated with the project outcomes, and the development of new and novel software components that will be produced by the technical workpackages. Moreover, all these development tasks will be carried out by a distributed team of developers from different institutions. Hence, Wf4Ever needs both appropriate means to support the development tasks, and practical guidelines to be followed by the development team.

In this document, we described the software development infrastructure that we have deployed in order to support the development tasks in Wf4Ever. This infrastructure includes different tools and recommended technologies supporting the following tasks:

- Source control management (Subversion - SVN)
- Issue management (JIRA)
- Build process (Maven)
- Continuous integration (CruiseControl)
- Testing (JUnit, JMeter, etc.)
- Software releases (CruiseControl, Maven, JIRA)

For the selection of these tools, we considered their suitability to support an agile software development methodology and the collaboration of a distributed team of developers. We also considered the software development tools used by each partner as well as the underlying technologies of the software components from each partner that may be integrated in the reference implementation.

Additionally, this document also described the most important guidelines related to software development practice, to be followed in Wf4Ever. In particular, we discussed good practices for using a source code repository, issue management, build process management and release management. We also provided a brief summary of coding guidelines to be followed in Wf4Ever, including general rules of good practice and recommendations to assure code readability and efficient exception handling and logging.

The description of the deployed software infrastructure and used guidelines will be maintained in the project wiki in order to reflect future updates due to emerging requirements during the project lifetime.

Finally, this document presents the inventory of software development tools and software components from each of the technological partners.

A Inventory of Software Development Tools

In this appendix we present the inventory of tools used by Wf4Ever partners for software development. To collect this inventory, each partner doing software development provided a description of the tools they use and how such tools are used (e.g., for what purpose, with what technologies). In particular, for each tool, each partner provided the following information:

- Name of the tool
- Version
- Usage
- Technologies for which it is suitable
- Runtime environment requirements
- Interface, e.g., technologies, protocols
- License
- Homepage URL
- Additional textual description
- Example

Table 1: Software Development Tools Used by Wf4Ever Partners (1/2)

Organization Name	Name of Tool	Version	Usage	Technologies for which it is suitable	Runtime environment requirements	Interface (technologies, protocols etc.)	License	Homepage URL	Additional Textual Description	Example
myGrid	Jira	4.11	Issue tracking, planning	Software development	Java (Tomcat bundled)	Web, RPC	Commercial, Free for open source projects	http://www.atlassian.com/software/j		http://www.mygrid.org.uk/dev/issues/
myGrid	Confluence	3.2.1	Wiki, documentation, notes	Software development	Java (Tomcat bundled)	Web, RPC	Commercial, Free for open source projects	http://www.atlassian.com/software/confluence/		http://www.mygrid.org.uk/dev/wiki/
myGrid	Hudson CI	1.367	Continuous testing	Java, Maven (integrated), Ruby (using shell scripts)	Java, Tomcat	Web, RPC to slaves, HTTP/SVN/Git triggers	New BSD (?)	http://hudson-ci.org/	Builds on every commited code change, sends email when code/test is broken. Nightly downloadable build, deploys to Maven snapshot repository. For releases, used for building final release and deploy to final repository	http://www.mygrid.org.uk/hudson/
myGrid	Crowd	2.0.4	Single sign-on	Jira, Confluence, Hudson, Wordpress, OpenID	Java (Tomcat bundled)	HTTP	Commercial, Free for open source projects			http://www.mygrid.org.uk/openid/users/stain
myGrid	Wordpress	2.8.6	Web pages, blogs	Home page, developer blog	PHP, Apache, mySQL	Web	GPL	http://wordpress.org/download/		http://www.mygrid.org.uk/dev/blog/ http://www.mygrid.org.uk/ http://www.taverna.org.uk/
myGrid	Google Apps		Email, Calendar	Email, documents, calendar	Hosted by Google	Web	Free (max 50 users)	https://www.google.com/a/		
myGrid	jProfiler	6.1	Debugging, memory/thread leaks, profiling	Java	Java	UI, remote connections to running Java applications/servers	Free for open source projects (if linking to product from project web page)	http://www.ej-technologies.com/products/jprofiler/overview.		
myGrid	Apache Maven	3.0.1	Building, testing, packaging	Java	Java	Command line, Eclipse plugin (m2eclipse)	Apache License, version 2.0	http://maven.apache.org/	Powerful build and dependency management system for Java	
myGrid	Google Code		Subversion repository, Mercurial, Download mirror	Subversion, Mercurial, (simple) wiki/issue tracker	Hosted by Google	Web, SVN	Free to use	http://code.google.com/		http://code.google.com/p/taverna/source/browse/taverna
myGrid	Launchpad		Download site	Download, Bazaar, Issue tracking	Hosted by Canonical Ltd (Ubuntu people)	Web	Free to use	https://launchpad.net/taverna/	Really fast download speeds	
myGrid	Eclipse	Helios (3.6.1)	Java development (IDE), UML	Java, Ruby, XML, UML, OSGi	Java	UI		http://www.eclipse.org/	We only use Eclipse for Java development, building and dependencies done by Maven and m2eclipse plugin	http://www.mygrid.org.uk/dev/wiki/display/developer/Checking+
myExperiment	Fogbugz	5.0.31	Issue tracking, planning	Software development	PHP, Apache, MySQL	Web	Commercial	http://www.fogcreek.com/FogBugz/	Plan to switch to Jira	http://bugs.myexperiment.org/
myExperiment	MediaWiki	1.12.0	Wiki, documentation, notes	software development	PHP, Apache, MySQL	Web	GPL version 2	http://www.mediawiki.org/		http://wiki.myexperiment.org/index.php/Main_Page
myExperiment	Nagios	2.10-5	Service monitoring		linux	Web	GPL	http://www.nagios.org/		
myExp/BioCat	RubyForge		source control, mailing lists, issues tracker			Web, SVN, Git		http://rubyforge.org/		
myExperiment/myGrid	SOLR	1.4.1, 1.3.x	search				Apache	http://lucene.apache.org/solr/		
myExperiment/myGrid	mySQL	5.0.x/5.1.x								
myExperiment/myGrid	Apache HTTPD	2.2.9-1								
myExperiment/myGrid	Ruby	1.8.x, 1.9.x								
myGrid	JRuby	1.5.x	Ruby on JVM							
myExperiment/myGrid	Rails	1.2.6, 2.3.x, 3.x								
myExperiment	Mongrel	1.1.5	Ruby App Server		Ruby					
myExperiment	Mongrel Cluster	1.0.5	Ruby App Server		Ruby					
myGrid	Phusion Passenger (mod_rails)	2.x	Ruby App Server		Ruby					
myGrid	Git	1.7.x	Version control							
myGrid	Github									
myExperiment/myGrid	VirtualBox		Deployment, development, distribution							
myExperiment	4store	1.04	RDF database	RDF, SPARQL	64-bit Fedora Core 13		GPLv3	http://4store.org/	Must be 64bit	
myExperiment	Raptor	1.4.21	RDF library	RDF, SPARQL			LGPL/Apache 2	http://librdf.org/raptor/		
myExperiment	Rasqal	0.9.20	RDF Query library	RDF, SPARQL			LGPL/Apache 2	http://librdf.org/rasqal/		
myExperiment/myGrid	RubyGems.org		Ruby GEM distribution					http://rubygems.org		
myGrid	Apache CXF	2.2.9	SOAP/REST web services	SOAP, REST	Java					

Table 2: Software Development Tools Used by Wf4Ever Partners (2/2)

Organization Name	Name of Tool	Version	Usage	Technologies for which it is suitable	Runtime environment requirements	Interface (technologies, protocols etc.)	License	Homepage URL	Additional Textual Description	Example
myGrid	memcached		Data Cache Server							
myGrid	Python	2.5+	Service testing scripts for BioCatalogue							
myGrid	PHP	<5.3								
myGrid	SourceForge		Mailing lists	Nothing else these days						
myGrid	Mercurial									
myExperiment	Dropbox		Content storage				not open source	https://www.dropbox.com/		
myGrid	delayed_job		Background processing	Ruby/Rails	Ruby					
myGrid/myExperiment	Google Analytics		Web traffic analysis	Web	Hosted by Google		Free			
myGrid	RPX Now		SSO delegation using OpenID/Facebook/T Apps		Hosted by Janrain			https://rpxnow.com/		
UPM	Nexus	1.8.x	Management of maven repositories.	Maven	Java JRE or Tomcat	Web interface	GPL	http://nexus.sonatype.org	Useful for publishing maven repositories of project releases. Also it is good for managing JAR third party dependencies.	
UPM	ReviewBoard	1.0.7	Code reviews		Python	Web interface for reviewing and command line interface for submitting reviews. REST	MIT	http://www.reviewboard.org/		http://reviews.alexdeleon.name
UPM	Trac	0.12b	Issue management, planning, project wiki			Web interface	BSD	http://trac.edgewall.org/		http://trac.alexdeleon. name/geolinkeddata
myGrid	Java	1.6								
PSNC	JDK	1.6	Code compilation and running			CLI	Proprietary	http://www.oracle. com/technetwork/java/index. html		
PSNC	Maven	1.1	Code compilation, dependencies management, project website generation, project building, ...	Java	Java	CLI	The Apache Software License, Version 2.0	http://maven.apache.org/		
PSNC	CruiseControl	2.8.4	Continuous integration, releases		Java	Web interface	BSD-style license	http://cruisecontrol. sourceforge.net/		
PSNC	Apache Subversion	1.5	Code versioning			HTTP	Apache License, Version 2.0	http://subversion.apache.org/		
PSNC	JIRA		Issue tracking		Java	Web interface + WS	Commercial	http://www.atlassian. com/software/jira/		
PSNC	Mediawiki		Internal wiki		PHP	Web interface		http://www.mediawiki. org/wiki/MediaWiki		
PSNC	Confluence		Users community wiki		Java	Web interface + WS	Commercial	http://www.atlassian. com/software/confluence/		
PSNC	Apache JMeter		Performance testing		Java	GUI	Apache License, Version 2.0	http://jakarta.apache. org/jmeter/		
PSNC	JUnit	4.x	Unit testing	Java	Java	API	Common Public License - v 1.0	http://www.junit.org/		
PSNC	PostgreSQL	8.x	Database for test instance			JDBC, SQL	BSD-style license	http://www.postgresql.org/		

B Inventory of Software Components

In this appendix we present the inventory of software components, relevant for the development of Wf4Ever reference implementation, that have been developed previously or that are being developed by Wf4Ever partners. To collect this inventory, each partner having potential software components that could be used/integrated in the reference implementation provided a short description of each component. In particular, for each component, each partner provided the following information:

- Name of the component
- Version
- Date
- Provided functionality
- Examples of use
- Target, e.g., research, pilot, production
- Development status, e.g., alpha, beta, stable
- Technologies used
- Development tools used
- Runtime environment requirements
- Interface, e.g., technologies, protocols
- License
- Homepage URL
- Source code location
- Additional textual description

Table 3: Software Components from Wf4Ever Partners

Organization Name	Name of Component	Version	Date	Provided functionality	Examples of use	Target (Research/ Pilot/ Production)	Development status (alpha, beta, stable)	Technologies used	Development tools used	Runtime environment requirements	Interface (technologies, protocols etc.)	License	Homepage URL	Source code location	Additional Textual Description
myGrid	Taverna Workbench	2.2.1	2010-07-27	Workflow editing and execution	http://www.youtube.com/watch?v=ZhDtn8amiAY	Production	stable	SOAP, REST, R, etc	Java, Maven, Spring, Raven	Java 6	Swing UI	LGPL 2.1	http://www.taverna.org.uk/download/workbench/	http://taverna.googlecode.com/svn/taverna/	
myGrid	Taverna command line	2.2.0	2010-07-07	Workflow execution	http://taverna.knowledgeblog.org/2010/12/13/taverna-command-line/	Production	stable	SOAP, REST, R, etc	Java, Maven, Spring, Raven	Java 6	Command line	LGPL 2.1	http://www.taverna.org.uk/download/command-line-tool/	http://taverna.googlecode.com/svn/taverna/	
myGrid	Taverna Server	2r1	2010-07-30	Workflow execution	http://www.taverna.org.uk/documentation/taverna-2-x/server/	Production	alpha	REST/SOAP	Java, Maven, JAXB	Java 6, Taverna Command line, Tomcat	SOAP/REST	LGPL 2.1	http://www.taverna.org.uk/download/server/	http://taverna.googlecode.com/svn/taverna/pr	
myGrid	Taverna Server Web Demonstrator	?	2010-07-30	Workflow execution frontend	http://heater.cs.man.ac.uk:4567/	Pilot	alpha	REST	Ruby	Ruby, Taverna Server	Web	New BSD	http://www.taverna.org.uk/download/server/	https://github.com/myGrid/Tavern	
	Taverna Server Ruby client		0.5.3		http://rubydoc.info/gems/t2-server/0.5.3/frames	Production	beta	REST	Ruby	Ruby	REST	New BSD		https://github.com/myGrid/t2-server-gem	
myGrid	Taverna Server Portlet demonstrator	?	2010-09-01	Portlet for workflow execution and data viewing	http://heater.cs.man.ac.uk:4040/	Production	alpha	REST	Java, Portlet JSR-168, JSP	Java, Tomcat, Portlet container (LifeRay)	Web portal		http://www.mygrid.org.uk/dev/wiki/display/portals/Deliverable+2.1.1+Enactment+of+Taverna+Workflows+via+a+Portal	http://taverna.googlecode.com/svn/unsorted/p	
myGrid	Taverna provenance API	?	2010-08-24	APIs for querying Taverna provenance	http://www.mygrid.org.uk/dev/wiki/display/provenance/Taver	Research	alpha	Java, RDF, SQL	Java, Maven	Java	Java API, Command line	LGPL 2.1	http://www.mygrid.org.uk/dev/wiki/display/provenance/Provenance+Management+in+Taverna	http://mygrid-labs.googlecode.com/svn/provenanc	
myGrid	SCUFL2	0.1	2011-02-01	Workflow language (API, XML/RDF)	https://github.com/myGrid/scufl2/blob/master/scufl2	Production	developing	Java, RDF	Java, JAXB	Java	Java API, SparQL, RESTful	New BSD (?)	http://www.mygrid.org.uk/dev/wiki/display/developer/2010-07+SCUFL2	http://github.com/myGrid/scufl2	
myGrid	Taverna Platform	?	2011-02-01	API for workflow execution/monitoring			developing	Java, OSGi	Java, Equinox OSGi, Spring	Java		LGPL 2.1		http://taverna.googlecode.com/svn/unsorted/p	
myGrid	Taverna 3 Workbench (next gen workbench)		2011-02-01	Workflow editing and execution		Production	developing	Java, OSGi, Eclipse Rich Client Platform (RCP), Graphics Editing Framework (GEF) and Standard Widgets Library (SWT)	Java, Eclipse RCP, Taverna Platform	Java, Equinox	SWT based GUI	LGPL 2.1 (?)	http://www.mygrid.org.uk/dev/wiki/display/TNG/Home	http://mygrid-labs.googlecode.com/svn/taverna-tng/trunk/taverna-workbench/	
myGrid & SysMO-DB	SEEK	0.8		Asset catalogue/Yellow Pages	http://www.sysmo-db.org/demo	Production	beta	Ruby on Rails 2	Ruby on Rails, C++, Assembler	Ruby	Web, REST	New BSD	http://www.sysmo-db.org/seek	http://code.google.com/p/sysmo-db/	
myGrid & SysMO-DB	Rightfield	0.12		Semantic markup of spreadsheets		Production	beta	Java, RDF	Java	Java	Swing UI	New BSD	http://www.sysmo-db.org/rightfield	https://github.com/myGrid/RightFi	
myGrid & EBI	BioCatalogue			Discovery and annotation of biological web services	http://www.biocatalogue.org/	Production	beta	Ruby, Rails 2, Java, Python, PHP, MySQL, memcache, Solr	Ruby on Rails 2	Ruby, Java	HTML, JS, REST (XML & JSON)	New BSD	http://www.biocatalogue.org/	http://rubyforge.org/scm/?group_id=6901	
Southampton, myGrid, Oxford, eResearch South	myExperiment			Discovery and sharing of scientific workflows and data	http://www.myexperiment.org/	Production	stable	Ruby on Rails 1	Ruby on Rails 1	Ruby, Java, PHP	HTML, XML, RDF, REST, OpenID, OAuth	New BSD	http://www.myexperiment.org	http://rubyforge.org/scm/?group_id=3801	
myGrid	Methodbox	Round 5		Finding/sharing datasets (surveys, etc)	http://www.methodbox.org/surveys	Production	stable	Ruby on Rails 2	Ruby on Rails 2	Ruby, Java	Web	New BSD	http://www.methodbox.org/	http://code.google.com/p/methodbox/s	
myGrid	Utopia Documents	1.1.1		View scientific papers with live data visualisations and metadata	http://getutopia.com/documents/quick_start.php	Research	beta(?)	PDF	QT, Python, C++	Linux/Windows	QT UI	Free to use	http://getutopia.com/documents/	Not open source	
myGrid	Taverna Workflow Components Library		2011-02-01	Publish/find wf components	https://rpxnow.com/	Production	developing	Ruby, Rails 3, Java, MySQL, mongo-db, memcache, Solr	Ruby on Rails 3	Ruby, Java	HTML, JS, REST (JSON)	New BSD			
SOA4All & iSOCO	kAnalytics			(Linked Data) graph visualization through SPARQL	http://soa4all.isoco.net/kAnalytics/about/	Pilot	beta	Java, RDF, jPowerGraph	Java, Maven	Windows, Java	Web, REST, SPARQL	Apache 2.0	http://soa4all.isoco.net/kAnalytics/about/		
PSNC	dLibra Digital Library Framework - Metadata service	5.0.0		Metadata storage and digital objects' structure management.	http://www.wbc.poznan.pl/dlibra	Production	stable	Java SE, Java RMI, Java Servlets, Swing, Lucene, Hibernate, JMX, ...	Java 1.6, Maven 1.x	Any OS with Java 1.6, Optionaly MySQL, PostgreSQL or Oracle DB	Java RMI, JMX, partially HTTP+XML	Open-source	http://dlibra.psnc.pl/	Internal PSNC repository	
PSNC	dLibra Digital Library Framework - Content service	5.0.0		Digital objects' content storage.	http://www.wbc.poznan.pl/dlibra	Production	stable	Java SE, Java RMI, Java Servlets, Swing, Lucene, Hibernate, JMX, ...	Java 1.6, Maven 1.x	Any OS with Java 1.6, Optionaly MySQL, PostgreSQL or Oracle DB	Java RMI, JMX, partially HTTP+XML	Open-source	http://dlibra.psnc.pl/	Internal PSNC repository	
PSNC	dLibra Digital Library Framework - Indexing service	5.0.0		Search index building - on descriptive metadata and textual content.	http://www.wbc.poznan.pl/dlibra	Production	stable	Java SE, Java RMI, Java Servlets, Swing, Lucene, Hibernate, JMX, ...	Java 1.6, Maven 1.x	Any OS with Java 1.6, Optionaly MySQL, PostgreSQL or Oracle DB	Java RMI, JMX, partially HTTP+XML	Open-source	http://dlibra.psnc.pl/	Internal PSNC repository	
PSNC	dLibra Digital Library Framework - Search service	5.0.0		Searching in the search index.	http://www.wbc.poznan.pl/dlibra	Production	stable	Java SE, Java RMI, Java Servlets, Swing, Lucene, Hibernate, JMX, ...	Java 1.6, Maven 1.x	Any OS with Java 1.6, Optionaly MySQL, PostgreSQL or Oracle DB	Java RMI, JMX, partially HTTP+XML	Open-source	http://dlibra.psnc.pl/	Internal PSNC repository	
PSNC	dLibra Digital Library Framework - User service	5.0.0		Users and content authorization management	http://www.wbc.poznan.pl/dlibra	Production	stable	Java SE, Java RMI, Java Servlets, Swing, Lucene, Hibernate, JMX, ...	Java 1.6, Maven 1.x	Any OS with Java 1.6, Optionaly MySQL, PostgreSQL or Oracle DB	Java RMI, JMX, partially HTTP+XML	Open-source	http://dlibra.psnc.pl/	Internal PSNC repository	
PSNC	dLibra Digital Library Framework - Event service	5.0.0		Event-based asynchronous service communication.	http://www.wbc.poznan.pl/dlibra	Production	stable	Java SE, Java RMI, Java Servlets, Swing, Lucene, Hibernate, JMX, ...	Java 1.6, Maven 1.x	Any OS with Java 1.6, Optionaly MySQL, PostgreSQL or Oracle DB	Java RMI, JMX, partially HTTP+XML	Open-source	http://dlibra.psnc.pl/	Internal PSNC repository	
PSNC	dLibra Digital Library Framework - System service	5.0.0		Services registry for dLibra.	http://www.wbc.poznan.pl/dlibra	Production	stable	Java SE, Java RMI, Java Servlets, Swing, Lucene, Hibernate, JMX, ...	Java 1.6, Maven 1.x	Any OS with Java 1.6, Optionaly MySQL, PostgreSQL or Oracle DB	Java RMI, JMX, partially HTTP+XML	Open-source	http://dlibra.psnc.pl/	Internal PSNC repository	
PSNC	dLibra Digital Library Framework - Profile service	5.0.0		Users profie storage.	http://www.wbc.poznan.pl/dlibra	Production	stable	Java SE, Java RMI, Java Servlets, Swing, Lucene, Hibernate, JMX, ...	Java 1.6, Maven 1.x	Any OS with Java 1.6, Optionaly MySQL, PostgreSQL or Oracle DB	Java RMI, JMX, partially HTTP+XML	Open-source	http://dlibra.psnc.pl/	Internal PSNC repository	

References

[BMZ10] Marek Lewandowski Antoine Delvaux Tihana Zuljevic Candido Rodriguez Rade Martinovic Spass Kostov Ognjen Blagojevic Branko Marovic, Marcin Wrzos and Waldemar Zurowski. Gn3 software developer best practice guide 2.0. online: http://www.geant.net/Media_Centre/Media_Library/Media%20Library/GN3-09-186v4_Software_Developer_Best_Practices_Guide_2.0.pdf, 2010.

[Fow06] Martin Fowler. Continuous Integration. Technical report, ThoughtWorks, May 2006. Available at http://www.martinfowler.com/articles/continuousIntegration.html.

[Mac] Ian Macdonald. The unofficial ruby usage guide. online: http://www.caliban.org/ruby/rubyguide.shtml.

[McC06] Tim McCune. Exception-handling antipatterns. online: http://today.java.net/pub/a/today/2006/04/06/exception-handling-antipatterns.html, 2006.

[Mic] Sun Microsystems. How to write doc comments for the javadoc tool. online: http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html.

[Mic99] Sun Microsystems. Code conventions for the java tm programming language. online: http://www.oracle.com/technetwork/java/codeconv-138413.html, 1999.

[pro] Maven project. Introduction to repositories. online: http://maven.apache.org/guides/introduction/introduction-to-repositories.html.