Project Number	IST-2006-033789
Project Title	Planets
Title of Deliverable	Interoperability Framework – Release Report 1
Deliverable Number	IF/D3
Contributing Sub-project and Work-package	IF/2-D3, IF/3-D3, IF/4-D3, IF/5-D3
Deliverable Dissemination Level	External PP - Restricted to other programme participants
Deliverable Nature	Report
Date	30th June 2007
Author(s)	Ross King (ARC)

Abstract

This Release Report briefly describes the vision, motivation, and benefits of the Interoperability Framework (IF) within the Planets project. Then it provides a description of each component developed within the Interoperability Framework over the last year. The description summarises the component, its technical characteristics, the implemented functions, development plans for the next phase, and known issues regarding the current implementation.

Keyword list:

Interoperability Framework, Components, Functionality

Contributors

Person	Role	Partner	Contribution
Carl Wilson	IF Developer	BL	Data Registry report Service Interface report
Klaus Rechert	IF Developer	ALUF	Monitoring and logging report
Thomas Krämer	IF Developer	UzK	Administration Interface report
Javier Botana	IF Developer	UzK	Service Registry report
Markus Reis	Technical Coordinator	ARC	Build Environment report
Rainer Schuster	IF/5 WP Lead	ARC	Workflow Design Tool report
Ross King	IF SP Lead	ARC	Integration and summary

EXECUTIVE SUMMARY

This document is the so-called “Release Report” that should accompany the following external Planets software prototype Deliverables:

IF/2-D3 Interoperability Infrastructure Prototype Implementation
IF/3-D3 Service Interface Prototype Implementation
IF/4-D3 Registry Framework Prototype Implementation
IF/5-D3 Orchestration and Choreography Prototype Implementation

This Release Report briefly describes the vision, motivation, and benefits of the Interoperability Framework (IF) within the Planets project. Then it provides a description of each component developed within the Interoperability Framework over the last year. The description summarises the component, its technical characteristics, the implemented functions, development plans for the next phase, and known issues regarding the current implementation.

TABLE OF CONTENTS

1. Introduction 5
1.1 Vision 5
1.2 Motivation 5
1.3 Benefits of the Interoperability Framework 5
2. IF/2-D3: Interoperability Infrastructure Prototype Implementation 7
2.1 Build Environment 7
2.2 Administration Interface 7
2.3 Monitoring and Logging 8
3. IF/3-D3: Service Interface Prototype Implementation 10
3.1 Service Interface 10
4. IF/4-D3: Registry Framework Prototype Implementation 11
4.1 Service Registry 11
4.2 Data Registry 12
5. IF/5-D3: Orchestration and Choreography Prototype Implementation 14
5.1 Workflow Design Tool 14

1. Introduction

1.1 Vision

Two years from today

A library is interested in the topic of preservation planning, and discovers the PLANETS website online. The library administrator is able to download a single (platform and operating system independent) PLANETS Installation Package to their local system. When she double clicks on this package, a local PLANETS Instance is installed.

The administrator now double-clicks on the PLANETS icon to start the local server, then opens a web browser and points to the local host, and reaches the PLANETS administration console. From this point she can create user accounts and deploy services, as well as browse the local data repositories and service registry (a number of characterization and migration services are included in the download package and are automatically deployed when the instance is started).

The administrator creates a user account for her colleague, the archivist, who is then able to log in to the server from his own web browser (assuming the local network is properly configured). The archivist sees a number of links to various PLANETS applications on his starting page: to the PLANETS Testbed Application, the PLANETS Preservation Planning Application, and the PLANETS Workflow Design Tool. There are also links to external services including JHove and PRONOM.

1.2 Motivation

There are a number of functions that all (or nearly all) software applications commonly need. These include:

- Data persistence
- User management
- Authentication and Authorization
- Monitoring, Logging, and Messaging
- Workflow definition and execution

This last item may require some additional comment. Several of the Planets components, such as the Testbed and Preservation Planning, require the system to execute a user-defined sequence of steps. Such a sequence is called a workflow. The component that executes a workflow is referred to as a workflow engine. A workflow design tool may help an end-user to define an appropriate sequence of steps chose from a library of available services.

There are some non-functional requirements on the infrastructure, including:

- Robustness
- Scalablablility
- Distribution

The purpose of the the Sub-project is to develop the Interoperability Framework software components that provide these commonly required functions while meeting the non-functional requirements.

1.3 Benefits of the Interoperability Framework

Efficiency

If the above mentioned components are only developed once, rather than multiple times, then the PLANETS Sub-projects and their applications can concentrate on their specific process logic and will have more time and resources to do so.

Also, when packaging the PLANETS software, the number of components will be reduced; for example, because the IF provides a single database for all components, only one database need be installed.

Interoperability

By using a common framework, the Planets software components remain interoperable. For example, by using Enterprise Java Beans (EJB), the IF ensures that communication between distributed components is optimized, either through local API calls, or Remote Method Invocation (RMI).

By enforcing Web Service standards, the IF supports access to remote and distributed third-party characterization and migration services.

2. IF/2-D3: Interoperability Infrastructure Prototype Implementation

2.1 Build Environment

2.1.1 Description

The Software Development Environment supports distributed teams located all over Europe in the development of Planets software. All tools and communication channels are integrated into and accessible from one platform, resulting in a low entry barrier. The core of the Software Development Environment is a Software Build Environment that automatically and continuously builds the latest software artefacts, runs tests and applies quality metrics. The source code and document management system provides a sound foundation. It includes versioning, branching, and concurrent manipulation. Bug, To-Do and feature trackers in combination with mailing lists and discussion forums create a helpful tool set in order to support knowledge transfer within the consortium.

The open source JBoss J2EE application server has been chosen as the main component of the infrastructure to support the tools and services to be built as part of the project. A pre-configured instance of JBoss has been prepared and included in the build environment.

2.1.2 Technical Characteristics

The Software Build Environment is based on GForge – a web-based portal – including various plugins. The project makes use of Subversion (SVN) for Source Code Management, a Document Management System for documentation and other project files, a WIKI, as well as item trackers for bugs, To-Do lists and feature requests. We use version 4.5 of GForge. Software builds are controlled by orchestrated ANT (v1.6.5+) scripts and automated (e.g. nightly builds) by CruiseControl (v2.6). Software quality will be ensured through software tests based on the JUnit Framework and applied software metrics (e.g. code coverage of the available tests) using tools like EMMA or Clover, and FindBugs.

2.1.3 Functionalities Implemented

The GForge is fully set up and working, including the SVN, the Document Management System, discussion forums, mailing lists, and item trackers. The Build Environment currently includes some basic ANT scripts (in order to build Java source code) and provides a preconfigured and packaged JBoss Application container in order to run the built software artefacts.

2.1.4 Development Plans

The set of ANT scripts should be more comprehensive and elaborate. The Software Quality Assurance Tools like EMMA, Clover, and FindBugs should be established. CruiseControl should be configured to automate continuous builds and present an up-to-date overview of the build status to interested developers and users.

2.1.5 Known Issues

It is planned to use Java 1.6 as the standard version of Java for the project. However, currently there is an incompatibility between the JBoss web services libraries and libraries included in Java 1.6. For now, we are using Java 1.5 while a solution to the problem is being investigated.

2.2 Administration Interface

2.2.1 Description

The basic task of the administration interface (AdminUI) is user management and role assignment.

The administration interface is a central application that performs generic tasks such as user management and assignment of roles to users. Some functions within the Planets project can only be performed by authorized staff. Therefore, the assignment of roles to users is crucial and affects all components in Planets.

The AdminUI may serve as an example for the Planets partners that need fine-grained access control in their applications.

2.2.2 Technical Characteristics

The administration interface is a web application called AdminUI, i.e. it will be found at "yourhost:port/AdminUI" at your Planets instance. This web application makes use of the acegi security framework for the purposes of authorization and authentication. User information (address, email, password etc.) is stored in a RDBMS.

Most modules in Planets go beyond the access control model and need to determine whether a user is authorised to use of certain functions. All three types of security (protocol, path and method based security) have been implemented in the first prototype of the AdminUI.

2.2.3 Functionalities Implemented

Users may apply for an account. The administrator assigns roles (service provider, for example) to users after their successful registration. These roles will then determine the pages and functions that a user can access and execute.

2.2.4 Development Plans

Currently, the Planets Single-Sign-On Server (PSSOS) uses username and password to perform an authentication. In future versions, it might be a requirement to switch from username/password to client side x509 certificates. PSSOS supports client side certificates, but is not yet configured to do so.

2.2.5 Known Issues

The use of client side certificates implies a certification authority (CA), against which client (and server) side certificates are validated. Planets does not presently have such a service. It must be decided, if we want to establish our own certification authority or if we can use an existing one from related projects.

2.3 Monitoring and Logging

2.3.1 Description

The Planets logging service provides a shared logging service. It provides well defined interfaces for collecting and distributing log messages and makes use of the Apache Log4J library. Log messages can be accessed via email or web services. Log messages for defined log-levels can also be passed to a Notification Manager where a remote user can register for monitoring information.

A Notification Manager provides an interface for other services to register to receive defined log messages. The Notification Manager might, for example, monitor the progress of workflows and notify remote clients with appropriate access rights.

2.3.2 Technical Characteristics

The Monitoring and Logging component has the following characteristics:

- The PlanetsLogger implements the Apache Commons Log interface and uses Log4J as its backend.
- The PlanetsLoggerConfigurator class allows administrators to add configuration directives at runtime.
- The PlanetLoggerInit servlet initializes the logging infrastructure at server start-up.
- Using Log4J allows the usage of some standard Appender Classes like File, Console, Socket, Syslog, NTEventLog, SMTP.
- The logging front end enables administrative users to set log levels and monitor log tails.
- The NotificationManager uses JMS for message handling and passing but might also use email / SMS / etc for delivering messages to subscribers.

2.3.3 Functionalities Implemented

The following Monitoring and Logging functionalities have been implemented:

- PlanetsLogger class provides logging channels with eu.planets_project prefix. A servlet loads an initial configuration file at server start-up time.
- A logging monitor / administration interface allows administrative users to modify log-levels and monitor log-tails of all logging channels with eu.planets_project prefix.
- All log messages for eu.planets_project (root)loggers are written to a file and console. This was statically configured in the planets-log4j.xml file. Additional Loggers can be configured at runtime via PlanetsLoggerConfigurator.

2.3.4 Development Plans

The following functionalities will be developed in the next project year:

- Notification Manager
- security (connect to existing sign-on back end)
- custom log levels
- definition of logging granularity
- inclusion of additional output channels

2.3.5 Known Issues

At present, the Logging front-end uses plain JSP/HTML pages instead of Java Server Faces (JSF); this will be migrated for the first IF release (M15).

3. IF/3-D3: Service Interface Prototype Implementation

3.1 Service Interface

3.1.1 Description

The Service Interface Prototype provides the connection between the Planets Interoperability Framework and exterior real world software components. The aim is to provide a web-service based standard which can wrap the functionality of external components in a Planets friendly manner.

The Service Interface allows an external component to interact with the standard Planets Components such as the Data Registry and Logging Mechanism. The Prototype Service wraps a third-party tool (from Microsoft) for the migration of proprietary data formats, and demonstrates the integration of varying operating systems through Web Services as well as the Planets distributed service-oriented architecture.

3.1.2 Technical Characteristics

The Service Interface is presented as a web service interface which can be called by the Planets BEPL Workflow Engine. Initially this service will be hosted within the JBoss instance used to run the Planets Interoperability Framework.

3.1.3 Functionalities Implemented

The Service Interface is itself not a functional component; rather, it provides the glue that connects external components to the Planets Interoperability Framework. It provides a standard template for communication with the Data Registry, and Logging components.

3.1.4 Development Plans

Possible interfaces to be developed include:

- Process control (Execute, Pause, Resume Cancel, etc.)
- Transaction control (Commit, Rollback / Compensation, etc.)
- Messaging

3.1.5 Known Issues

It was not possible to address process control issues for the first prototype; this issue will be addressed in the coming project year.

4. IF/4-D3: Registry Framework Prototype Implementation

4.1 Service Registry

4.1.1 Description

The Planets Service Registry will enable Users and Service Providers to look up and publish information about Planets Services, as well as System Administrators to manage information about Planets Services. This information can be used to dynamically select and invoke services as simple services, as well as to reuse them as part of choreographed complex workflows. In order to find services, the Planets Service Registry contains information about the services and the publishers that provide them. Moreover, it provides an extensible, schema- or ontology-based service categorization mechanism that allows the Service Registry to be queried. In order to invoke or reuse services, the Planets Service Registry will contain information about services, including its interfaces, operations and parameters.

4.1.2 Technical Characteristics

The Planets Service Registry consists of two main components: the Service Registry itself that functions as a server and processes requests from clients, and the Service Registry User Interface that is integrated within the Administration user interface. The Service Registry is a Web application implemented in the Java programming language. The code used was taken from the Apache jUDDI project, an open-source Java-UDDI-version-2.0-compliant implementation available under the Apache License. (UDDI is the Universal Description, Discovery and Integration standard for web services). It is deployed as an independent application. The Service Registry User Interface is implemented as a Web application within the Administration User Interface, and provides the functionality to interact with the Service Registry. Java Server Faces (a framework for building web user interfaces) was used for the Web tier and JAXR (the Java API for XML Registries) as implemented by Apache Scout were used for the business logic tier. As it is a part of the Administration User Interface, it is deployed together with it.

4.1.3 Functionalities Implemented

The Planets Service Registry processes requests from clients (e.g. the Service Registry User Interface) and reads or writes the appropriate data from/to its backend. More precisely, it converts SOAP messages coming from UDDI 2.0-compliant clients into SQL statements and sends UDDI 2.0-compliant messages back to the clients (fulfilling the function of an object-relational mapping layer). The Planets Service Registry User Interface provides the Web forms and the business logic to browse, add, update, and delete items stored in the Service Registry. As it is part of the Administration User Interface, which shows or hides user interface items and so controlling the available functionality according to the specific role of the current user, the Service Registry User Interface makes use of this authorization system for the same purposes. A Planets User can only browse the Service Registry; a Planets Service Provider has the same privileges as a user but can additionally register his organization and publish/unpublish services. A Planets Administrator has all privileges.

4.1.4 Development Plans

Currently, the only possibility to discover services is to search by organization (wildcards are allowed) and to browse through their published services. This is only possible, if you know at least part of the organization's name. A more natural way would be to categorize the services thus allowing users to browse them by category. To this purpose, a Planets Service Taxonomy needs to be developed and stored within the Planets Service Registry backend. As the UDDI 2.0 standard already provides support for taxonomies, no changes to the Planets Service Registry are needed. The Planets Service Registry User Interface, however, would need to be modified in order to allow one to use the categorization system in discovering and publishing services.

Another useful feature would be a subscription mechanism: users of the Planets Service Registry can subscribe to specific services and are notified when these are changed. Additionally, a more general subscription mechanism could be used, notifying users about specific events (e.g. additions, updates, etc.).

4.1.5 Known Issues

Currently, when a user registers with the Planets Administration User Interface he is assigned the role of a Planets User. The Planets Administrator can change his role to Planets Service Provider if he wishes so. Now, in order to be able to publish services, the user has to be added as a provider to the backend of the Planets Service Registry. There is currently no way to do this via the API and needs to be done manually either as a SQL script or via the jUDDI console.

4.2 Data Registry

4.2.1 Description

The Data Registry provides storage and persistency services via a consistent API. Storage of and access to files and metadata is performed through the Data Registry. The Data Registry uses the Apache Jackrabbit implementation of the standard Java Content Repository API (as specified by Java Specification Request 170).

Binary data is stored by reference only to avoid the overhead of submitting and accessing it through the XML-oriented Jackrabbit.

4.2.2 Technical Characteristics

Data Registry

The data registry is implemented using Jackrabbit exposed through JNDI (Java Naming and Directory Interface) on JBoss. The full API is available as Java methods callable by IF components; a portion of the API is also exposed as a web service interface.

GUI

The GUI is web-based and implemented on the JSF1.2 specification using the Apache MyFaces implementation.

4.2.3 Functionalities Implemented

Data Registry

The data registry controls the Planets shared storage area, i.e. common network storage accessible by any IF component. Each registered user has their own storage area in which they can:

- Add content file references and associated metadata
- Save references to files created by workflow tasks
- Save metadata generated by workflow tasks
- Search content and create standard IF fileset messages using XQuery

GUI

The current GUI is a low-level graphical interface to the underlying Java Content Repository the Planets Data Registry is built on. Many functions offered are therefore not (yet) Planets specific.

- Browse content/data (by navigating through the tree representation)
- Create, Delete, and Modify content (e.g. by adding new content, modifying properties, etc.)
- Search content (by both XPath and SQL queries)

4.2.4 Development Plans

Data Registry

A prospective feature list for the Data Registry includes:

- Fine grained access control to files and metadata
- Export and Import of Data Registry contents to other user areas / IF instances
- Transaction control (Rollback/Commit)

- Versioning of data registry branches

GUI

The GUI will be extended by higher-level Planets-specific functions that allow Planets end users, for example librarians or archivists, (or other non-experts/administrators) to work with the DataRegistry GUI in a more intuitive task-related manner.

4.2.5 Known Issues

None.

5. IF/5-D3: Orchestration and Choreography Prototype Implementation

5.1 Workflow Design Tool

5.1.1 Description

The Workflow Design Tool (WDT) offers a graphical user interface to orchestrate Planets Services into a new Planets workflow. Workflows are modelled in a graphical notation and can then be exported as a workflow to the Workflow Execution Engine (WEE) in an XML format. The XML specifies the flow of the invoked services using W3C's Business Process Execution Language for Web Services (BPEL4WS). Once the WEE processes the Workflow, the WDT creates all relevant WSDL files and deploys the process as a Web Service on the JBoss Application Server's Web Service Container. Now the whole process acts like a simple Web Service Endpoint. The WDT builds upon the Open Source tool Eclipse BPEL designer in combination with other Open source plug-ins. During the development of the WDT, the IF/5 developers worked together with the Eclipse BPEL Designer development team, in order to contribute to the enhancements of this Open Source Tool.

5.1.2 Technical Characteristics

The Workflow Design Tool (WDT) is a standalone application based on the Eclipse Modelling Framework (EMF) and the Eclipse BPEL designer. It is extended by the Open Source Plug-in Eclipse Web Tools Platform (WTP), in order to retrieve a UDDI entry for certain services. The WDT instance is packaged in a folder and can be executed by running the executable Start-file. In order to deploy and execute the Planets workflows, a running JBoss Application Server including the jBPM BPEL extension are prerequisites.

5.1.3 Functionalities Implemented

The following functionalities are available in the first WDT prototype:

- Modelling a Workflow
    1. Create a new Planets Workflow Project
    2. Retrieve the Service Registry for Services
    3. Import the services to be invoked
    4. Design the Planets Workflow
- Deploying a Workflow
    1. Start the Deployment Wizard for instructions
    2. Run the ant-scripts for deploying to jBoss
- Monitoring a Workflow
    1. Monitoring all deployed workflows via the IF Web Application

5.1.4 Development Plans

Currently the Workflow Design Tool (WDT) is suitable for use by Orchestration experts. Although powerful, BPEL is complex and not quite mature, and so a certain level of Orchestration & Choreography knowledge of the Planets user is a prerequisite. Thus the WDT in the current development phase is called Expert WDT. For this reason it is planned to implement a more easily understandable and user-friendly workflow design tool over the next year. This design tool may be realized as a web application, where the Planets user can re-use specific workflow design patterns by adjusting only the data of such a pre-defined workflow. Furthermore the Expert WDT will be adjusted to enhancements made by the Eclipse BPEL designer Open Source community.

5.1.5 Known Issues

BPEL complexity and WDT usability are the primary issues; refer to the Development Plans above.