



Project no. 033572

## CASPAR

Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval

**Instrument:** Information Society Technologies

**Thematic Priority:** 2.5.10 Access to and preservation of cultural and scientific resources

# D3101:REPORT ON FRAMEWORK ARCHITECTURE



---

Document identifier:	<b>CASPAR-D3101-RP-0101-1_2</b>
Submission Date:	<b>03-02-2008</b>
Due Date:	<b>07-03-2008</b>
Work package:	<b>3100</b>
Partners:	<b>All Partners</b>
WP Lead Partner:	<b>ACS</b>
Document status	<b>DRAFT</b>

---

**Abstract:** The aim of this document is to define and describe the standards and methods that will be adopted in the SW development within CASPAR



**Delivery Type** Report  
**Author(s)** CASPAR Consortium

**Approval Summary** David Giaretta

**Keyword List**

**Availability**  PUBLIC

#### Document Status Sheet

Issue	Date	Comment	Author
0_1	11 09 2007	Initial draft	Marco Fulcoli
0_2	08 10 2007	Some additions	Marco Fulcoli
0_3	10 10 2007	Minor cut & paste corrections	Marco Fulcoli
0_4	19 11 2007	Released version with par. 4.4	Marco Fulcoli
1_0	01 02 2008	Integrate with Best Practice work	David Giaretta and Luigi Briguglio
1_1	15 02 2008	Minor corrections	David Giaretta, Luigi Briguglio and Marco Fulcoli
1_2	04 03 2008	Additional editing and expansion of some sections	Simon Lambert





### Project information

Project acronym:	<b>CASPAR</b>
Project full title:	<b>Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval</b>
Proposal/Contract no.:	<b>IST-2006-033572</b>

### Project Officer: Carlos Oliveira

Address:	<p>INFSO-E3 Information Society and Media Directorate General Content - Learning and Cultural Heritage</p> <p>Postal mail: Bâtiment Jean Monnet (EUFO 1167) Rue Alcide De Gasperi / L-2920 Luxembourg</p> <p>Office address: EUROFORUM Building - EUFO 1167 10, rue Robert Stumper / L-2557 Gasperich / Luxembourg</p>
Phone:	+352 4301 33052
Fax:	+352 4301 33190
Mobile:	
E-mail:	Carlos.Oliveira@ec.europa.eu

### Project Co-ordinator: David Giaretta

Address:	<p>STFC (formerly CCLRC), Rutherford Appleton Laboratory Chilton, Didcot, Oxon OX11 0QX, UK</p>
Phone:	+44 1235 446235
Fax:	+44 1235 446362
Mobile:	+44 (0) 7770326304
E-mail:	<a href="mailto:d.i.giaretta@rl.ac.uk">d.i.giaretta@rl.ac.uk</a>





## CONTENT

<b>1</b>	<b>INTRODUCTION.....</b>	<b>6</b>
1.1	PURPOSE OF THIS DOCUMENT.....	6
1.2	HOW TO READ THIS DOCUMENT.....	6
1.3	APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS .....	7
1.4	GLOSSARY.....	7
<b>2</b>	<b>FRAMEWORK REQUIREMENTS .....</b>	<b>9</b>
2.1.1	<i>Changes in scope.....</i>	9
<b>3</b>	<b>UNDERLYING DESIGN PRINCIPLES FOR PRESERVABILITY.....</b>	<b>11</b>
3.1.1	<i>Approaches.....</i>	11
3.2	COMMUNICATIONS PROTOCOLS .....	12
3.2.1	<i>Approaches.....</i>	12
3.3	DEPLOYMENT.....	12
3.3.1	<i>Approaches.....</i>	13
3.4	RESOURCE DISCOVERY .....	13
3.4.1	<i>Approach.....</i>	13
<b>4</b>	<b>INITIAL BEST PRACTICE .....</b>	<b>14</b>
4.1	PRINCIPLES.....	14
4.2	GUIDANCE FOR DEVELOPERS.....	16
	SERVER SIDE .....	16
4.2.1	<i>Step 1 – Define the component interface.....</i>	16
4.2.2	<i>Step 2 – Implement the component interface.....</i>	16
4.2.3	<i>Step 3 – Define the component skeleton.....</i>	17
4.2.4	<i>Step 4 – Deployment descriptor files.....</i>	18
4.2.5	<i>Step 5 – Generation of server artifacts .....</i>	18
	CLIENT SIDE.....	18
4.2.6	<i>Step 6 – Generation of client artifacts.....</i>	18
4.2.7	<i>Step 7 – Define client component.....</i>	18
	ADVANCED CLIENT SOLUTION: SERVICEFACTORY .....	19
4.3	NEXT STEPS .....	20
<b>5</b>	<b>FURTHER INVESTIGATIONS.....</b>	<b>21</b>
5.1	DATA EXCHANGE MECHANISM.....	21
5.1.1	<i>Motivation .....</i>	21
5.1.2	<i>Issues and approaches .....</i>	21
5.1.3	<i>Technologies .....</i>	21
5.2	MESSAGING MECHANISMS .....	21
5.2.1	<i>Motivation .....</i>	21
5.2.2	<i>Issues and approaches .....</i>	21
5.2.3	<i>Technologies .....</i>	22
5.3	COMMUNICATION IN A DISTRIBUTED ENVIRONMENT.....	22
5.3.1	<i>Motivation .....</i>	22
5.3.2	<i>Issues and approaches .....</i>	23
5.3.3	<i>Technologies .....</i>	24
5.4	REMOTE COMMUNICATION IN SECURE/INSECURE CHANNELS; .....	24
5.4.1	<i>Motivation .....</i>	24
5.4.2	<i>Issues and approaches .....</i>	24
5.5	FROM AN EMBEDDED TO A DISTRIBUTED SYSTEM .....	24
5.5.1	<i>Motivation .....</i>	25
5.5.2	<i>Issues and approaches .....</i>	25
5.6	PROCESS FLOW MANAGEMENT .....	28
5.6.1	<i>Motivation.....</i>	28
5.6.2	<i>Issues and approaches .....</i>	28





5.6.3	<i>Technologies</i>	28
<b>6</b>	<b>DEVELOPMENT SUPPORT TOOLS FOR CASPAR</b>	<b>30</b>
6.1	STANDARDS AND METHODS	30
6.1.1	<i>OO design and development</i>	30
6.1.2	<i>UML</i>	30
6.1.3	<i>Design</i>	30
6.1.4	<i>Development</i>	31
6.1.5	<i>Configuration Control</i>	31
6.1.6	<i>Development and Configuration Control Tools and Frameworks</i>	32
6.1.6.1	SVN	32
6.1.6.2	GFORGE	33
6.1.6.3	ANT	33
6.1.6.4	CruiseControl	34
6.1.6.5	MAVEN	34
6.1.6.6	SPRING	34
6.1.6.7	Spring-ws	36
6.1.6.8	Java API for xml web-services: JAX-WS	36
6.1.6.9	AXIS	36
6.1.6.10	Mule	37
6.1.7	<i>Tool collaboration and integration</i>	37
6.1.7.1	GForge & SVN	37
6.1.7.2	ANT & Maven	37
6.1.7.3	Spring & Mule	37
6.1.8	<i>Test Tools</i>	37
6.1.8.1	Verify & Validate	38
6.1.8.2	Test Level	38
6.1.8.3	Test case & Scenario	38
6.1.8.4	A possible testing cycle	39
6.1.8.5	Interfaces and Mock Objects	39
6.1.8.6	Techniques & frameworks	39
6.1.9	<i>Evolution Management</i>	40





# 1 INTRODUCTION

## 1.1 PURPOSE OF THIS DOCUMENT

The CASPAR framework is a software platform being implemented in the scope of the project that will allow the CASPAR components to interoperate. This is not only so that the case studies may be implemented, but to provide a lasting legacy from the project, by providing a flexible and customisable basis for future applications of the CASPAR components or their possible successors.

The purpose of this document is to bring together and summarise the work which has been undertaken to develop the framework within which the CASPAR components, and others, can work. It should provide a good understanding of how this framework is being implemented.

It will also provide an understanding of how we believe that we have taken into account the need for this kind of framework itself to be preservable.

## 1.2 HOW TO READ THIS DOCUMENT

The Framework Architecture has a number of aspects which this document tries to lay out in a logical fashion.

Section 2 collects together the requirements on the Framework from the CASPAR DoW, the Conceptual Model and the Component Architecture. These requirements are at high level rather than detailed, and can perhaps be better seen as expectations on what the framework should offer and how it should operate.

Since one of the most challenging requirements is the *preservability* of the Framework, in the sense of being able to cope with changes in technologies, Section 3 outlines the key ideas we have followed.

While we have very broad requirements, we also recognise the need to be able to rapidly prototype and demonstrate our work; therefore in Section 4 we lay out our current thinking on how best to use Web Services, which provide us with good sets of tools, a flexibility which suits our purposes, and is something which is currently widely supported.

Section 5 outlines a number of the other techniques which we are investigating as we add greater functionality in the Framework and in order to move beyond the initial Web Services.





### 1.3 APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

#### Applicable documents

[A1]	Description of Work, April 2006 ( <a href="http://www.casparpreserves.eu/Members/metaware/ReferenceDocuments/caspar-description-of-work/at_download/file">http://www.casparpreserves.eu/Members/metaware/ReferenceDocuments/caspar-description-of-work/at_download/file</a> )
[A2]	D4102 ( <a href="http://wiki.casparpreserves.eu/pub/Main/Deliverable4102/CASPAR-D4102_v0.2.doc">http://wiki.casparpreserves.eu/pub/Main/Deliverable4102/CASPAR-D4102_v0.2.doc</a> )
[A3]	Caspar guidelines;
[A4]	Caspar best practices;

#### Reference documents

[R1] CASPAR proposal, Sept 2005

### 1.4 GLOSSARY

[Ax]	Applicable Document
[Rx]	Reference Document
CASPAR	Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval
DoW	Description of Work
EC	European Commission
EPM	Executive Project Management
IPC	IP Coordinator
IST	Information Society Technologies
PACP	Partner Administrative Contact Point
PO	Project Officer
PPR	Project Progress Report
PQE	Project Quality Engineer
PTCP	Partner Technical Contact Point
R&D	Research and Development
SQE	Stream Quality Engineer
ST	Stream
TN	Technical Note
WP	Work Package
WPL	Work Package Leaders
Designated Community	An identified group of potential Consumers who should be able to understand a particular set of information. The Designated Community may be composed of multiple user communities. (OAIS definition)





Archival Information Package (AIP)	An Information Package, consisting of the Content Information and the associated Preservation Description Information (PDI), which is preserved within an OAIS. (OAIS definition)
Content Information	The set of information that is the original target of preservation. It is an Information Object comprised of its Content Data Object and its Representation Information. An example of Content Information could be a single table of numbers representing, and understandable as, temperatures, but excluding the documentation that would explain its history and origin, how it relates to other observations, etc. (OAIS definition)
Knowledge Base	A set of information, incorporated by a person or system, that allows that person or system to understand received information. (OAIS definition)
Representation Information	The information that maps a Data Object into more meaningful concepts. An example is the ASCII definition that describes how a sequence of bits (i.e., a Data Object) is mapped into a symbol. (OAIS definition)





## 2 FRAMEWORK REQUIREMENTS

Within the CASPAR project, Stream 2 is concerned with the development of components that provide key functionality in a system for digital preservation. This covers the components that relate directly to the OAIS functional model, as well as those that are related to access issues (DRM, authenticity, etc.). These components need to be integrated by means of a coherent framework that should allow their substitution by other components offering the same functionality. The development of such a framework is the job of Stream 3. The CASPAR project does not expect that the specific components developed in the project will live forever; indeed they might well have shorter lives than the digital material that they are helping to preserve. It is therefore essential that the components be integrated and managed within a framework that is able to handle them in a clean, coherent way.

A number of high-level requirements were expressed for the framework in the DoW [A1]. These arose from considering the basic expectations raised by lines of reasoning such as the one summarised above. These high-level requirements are:

- To provide a **reliable common infrastructure** that enables the building of services and applications that can be adapted to multiple areas
- **To be preservable** - it is expected that the architecture will leverage the many recent developments of Web Services and the GRID. Yet, even as one thinks about that, the difficulty of the challenge becomes clear; what would this project have been planning to use even five years ago – and how would that have survived the rise of Web Services? Preservability implies that the CASPAR framework can be layered on top of multiple underlying protocols
- To include the integration/customisation of a number of **basic but essential features** to build the CASPAR testbeds, such as: security, authentication, accounting, authorisation, monitoring, as well as the technologies to build systems that reflect a Service Oriented Architecture (SOA) style.
- To be relatively **easy to integrate** into existing systems
- To be designed in the light of **maximum flexibility** to simplify the integration of all technologies available.
- To allow the **monitoring and control** of the various preservation tasks and the updating of preservation procedures as long as new application exigencies emerge, and as technology evolution allows the implementation of more effective services.
- **To allow the re-use of existing components** in the project where valuable, in order to maximise the effectiveness of their integration and to harmonise them with the software infrastructure and with other components.
- To support (1) very-high volume, complex digital objects, oriented towards processing: Scientific testbed (2) dynamic interactive digital objects, oriented towards presentation and replay: Contemporary Arts testbed (3) virtual digital objects, spanning between processing and display, and able to overlap with each: Cultural Data Testbed.

The Framework will have to be compatible with the DoW [A1], the Component Architecture [A2], the CASPAR Guidelines [A3] and the CASPAR Best Practice [A4].

The management of software configuration will be decided and applied, with the objective of maintaining the full control of modules developed or integrated during the project lifetime.

### 2.1.1 Changes in scope

It is useful here to note that our views of the Framework have developed since the conception of the project and have changed in the following ways:

- Less formal workflow control
- More able to be tailored to individual repository systems





- Increased importance of preservability and separation of business logic from communications
- Increased variety of deployment scenarios
- Better appreciation of the choice of available technologies

These changes in scope are within the original conception of the framework but they affect the way in which we have approached its development.





### 3 UNDERLYING DESIGN PRINCIPLES FOR PRESERVABILITY

This section describes the underlying design principles which the CASPAR Framework will need to use. The foremost of these principles is preservability, which we discuss separately but which also permeates the other design aspects.

It would be fairly easy, using the toolkits which are available today<sup>1</sup> to produce a distributed system within which the CASPAR Key Components [D1301] can operate. However we are producing a system which will support digital preservation and so we should naturally give some thought to the preservability of the infrastructure which we expect others to depend upon, otherwise it could not be sustainable in the long term. Besides the preservability built into the design, there are clearly other aspects such as the longevity of the support for Key Components themselves; those considerations are not part of this document.

The preservability of our systems does not mean that we expect the software we produce to be unchanged over time. Instead we mean that our systems can survive changes in technologies without repeated wholesale re-design and re-writing of the CASPAR systems. For example if (or rather when) Web Services are replaced by the next generation of such services, the CASPAR Framework can be moved relatively easily to those new technologies. We believe this strategy must be adopted because the alternatives which might be proposed, namely maintaining Web Service implementations (the “Open Source” approach) or emulation of systems to keep the systems running (the “emulation” approach), are untenable. The “Open Source” approach would require a huge and unsustainable effort, the “emulation” approach would simply not work in a distributed system because it would require changes to archive systems which are outside our control.

#### 3.1.1 Approaches

The preservability requirements and the extreme asynchronous nature of the processes involved have several implications, summarised in the following table.

Issue	Approach
We cannot assume a single roll-out of all components. Interoperability must be maintainable over time as each component and the protocols used, evolves	The design of the interfaces of the Key Components has tried to be simple as possible and as implementation neutral as possible so that changes in lower layers can be implemented without changing the whole design.  (see also section 3.3)
Messages which are exchanged may not be acted upon until considerable time later, when components may have evolved.  Each component which can have such long-lived messages must be able to act as an archive of such messages in the sense that the information encoded in the message must be usable by the designated community in the future. This does not imply that the messages are to be kept indefinitely, merely that the messages must be able to survive changes in technologies before	Messages which need to be preserved must be treated as SIP (Submission Information Packages) i.e. they are pieces of information which are produced by someone (or something), then passed into an archive. An AIP (Archival Information Package) must be able to be produced from the SIP.

<sup>1</sup> A selection of which are described in the Appendix





being passed to the final recipient (designated community).	
The software must be maintained/evolved after the CASPAR project	<p>Any impediments to others playing an active role must be removed.</p> <p>One way of doing this is to ensure that the key interfaces are project independent e.g. package names for interfaces should be</p> <ul style="list-style-type: none"> <li>- info.preservedigital.*</li> </ul> <p>CASPAR's own implementations of these interfaces can be</p> <ul style="list-style-type: none"> <li>- eu.casparpreserves.*</li> </ul>
JAVA programming language may/will become obsolete.	<p>Wherever practical a Platform Independent Model (PIM) will be adopted in a UML tool, and an appropriate converter employed. However it is recognised that the business logic will be difficult if not impossible to capture completely in this way.</p>

## 3.2 COMMUNICATIONS PROTOCOLS

As has been mentioned above, although some form of Web Services provide the obvious communications protocol to use, we should be careful not to “hard-code” dependencies on Web Services throughout our software because it will make it extremely difficult if not impossible to survive the demise of Web Services.

The same considerations apply to the communications with the parameter systems and the logging system which is used to monitor the system.

### 3.2.1 Approaches

The aim is to ensure that the dependencies of the communications between Key Components and their clients are as isolated as possible.

The definition of the interfaces should be carefully considered. Although we cannot predict future communication protocols we can at least convince ourselves that our interfaces could be implemented fairly easily on several of the currently available such protocols e.g. RMI, CORBA as well as several flavours of Web Services.

The CASPAR implementations of the communications should be separated from the business logic implementations of the Key Component interfaces. Ideally, since we are using JAVA, the business logic should be implemented as *Plain Old Java Objects* (POJOs) and the communication aspects handled by a framework such as SPRING. Aspect Oriented programming techniques allows the communications and logging to be kept separate from the business logic. Selection of the application framework is described in the next section.

## 3.3 DEPLOYMENT

Clearly we should be able to deploy a complete system easily, with central monitoring and control of components, but we should also be support a number of more independent deployments as described in [D1301], without single points of failure.





Each component may evolve at different rates and the deployment mechanisms will also evolve.

### 3.3.1 Approaches

Design approaches such as the Enterprise Service Bus will be used.

For ease of demonstrations and roll-out of combinations of components, one of the common deployment systems will be used.

## 3.4 RESOURCE DISCOVERY

Although our initial implementation will have the locations of and communications protocols supported by the Key Components explicitly known nevertheless we should prepare for a more heterogeneous state.

### 3.4.1 Approach

During the initial development phase the limited number of services will be named explicitly in a number of properties files, and all components will use Web Services.

At the end of the development we aim to have a preservable resource discovery technique. While we have not fixed on this yet an obvious one would employ a number of special discovery nodes. As discussed in the Conceptual Model [D1201], we must accept that the network of resources may be fragmented i.e. not all resources will be discoverable by all others.





## 4 INITIAL BEST PRACTICE

### 4.1 PRINCIPLES

In this section there is a snapshot of the **CASPAR** Architecture Team's best practice guidance for developing the **CASPAR** Key Components. This guidance is aimed at two audiences:

- Developers of components within the project—though in fact the guidance has been formulated in conjunction with them and building on their experience in the project
- Future developers of components that will, in due course, substitute for the ones that now exist.

The best practice comes from the experiences analysed in the Best Practice document and is mainly based on the JAX-WS standard.

But it's important to remark that the **CASPAR** Key Components interfaces and implementations will have no dependency with the chosen development technology. In fact, in order to guarantee the "preservability" of the models and of the implementation (even if in part), the **CASPAR** Key Components interfaces will be just the "code generated by the UML models" and the implementations will be just the business logic implementations.

Dependencies to technology and underlying framework are, while (for the moment) we use a skeleton/stub approach, moved to the skeletons and stubs, as shown in the schema below.

In this case it's possible to assert that **CASPAR** Key Components implementation is "Technology Agnostic".

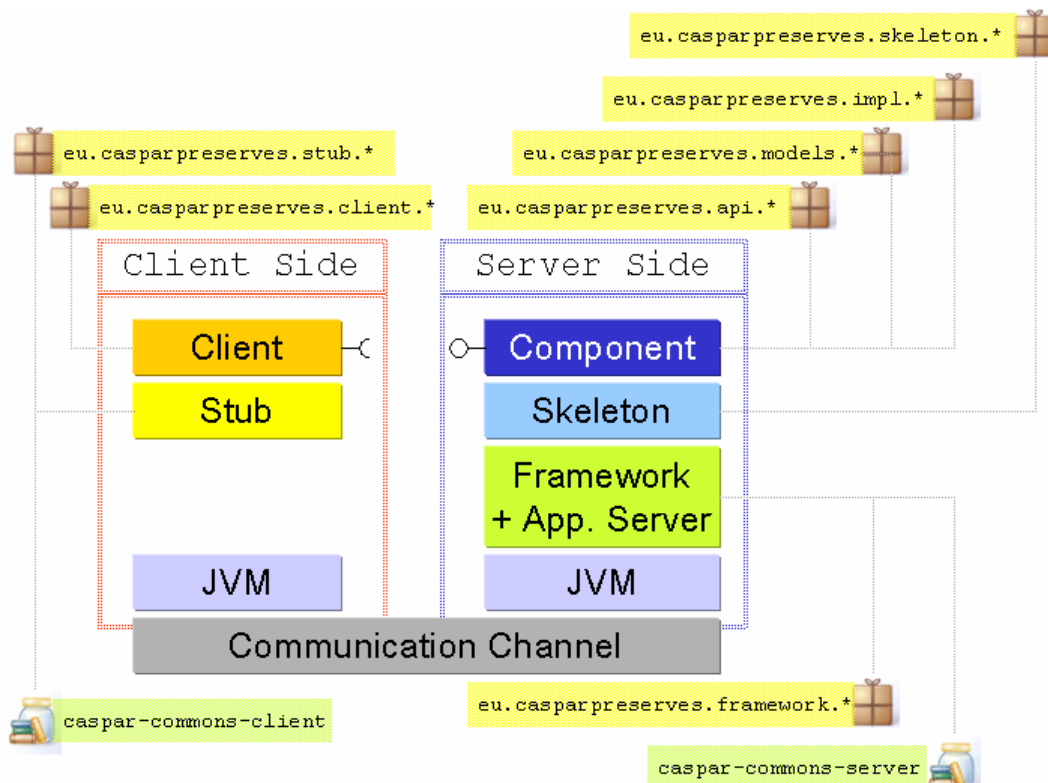


Figure 1 CASPAR Best Practice Layers

In this perspective, **CASPAR** Architecture Team identifies the following main packages for the **CASPAR** Key Components development:





- **eu.casparpreserves.api.\*** - package for the **CASPAR** Key Components interfaces, specified in [D1301] and their refined specifications. The code is generated by Enterprise Architect tool;
- **eu.casparpreserves.models.\*** - package for the **CASPAR** Key Components conceptual models (i.e. classes for parameters/data of operations of **CASPAR** Key Components interfaces), specified in [D1301] [D1201] and their refined specifications. The code is generated by Enterprise Architect tool;
- **eu.casparpreserves.impl.\*** - package for the implementation of the business logic of the **CASPAR** Key Components. No dependencies with the development technology and the underlying framework;
- **eu.casparpreserves.skeleton.\*** - package for the part of **CASPAR** Key Component implementation which interacts with the underlying framework and chosen technology;
- **eu.casparpreserves.framework.\*** - package for the common **CASPAR** functionalities which deal with registration, description and underlying framework interaction;
- **eu.casparpreserves.stub.\*** - package for the part of the client of the **CASPAR** Key Component which interacts with the underlying framework and chosen technology;
- **eu.casparpreserves.test.\*** - package for the **CASPAR** Key Component test, independent from the underlying framework and chosen technology;
- **eu.casparpreserves.client.\*** - package for the **CASPAR** Key Component which can be invoked from client applications. This package uses the generated `eu.casparpreserves.stub.*` artifacts.

The picture below shows the main packages, used by this best practice, with their relative classes and interfaces.



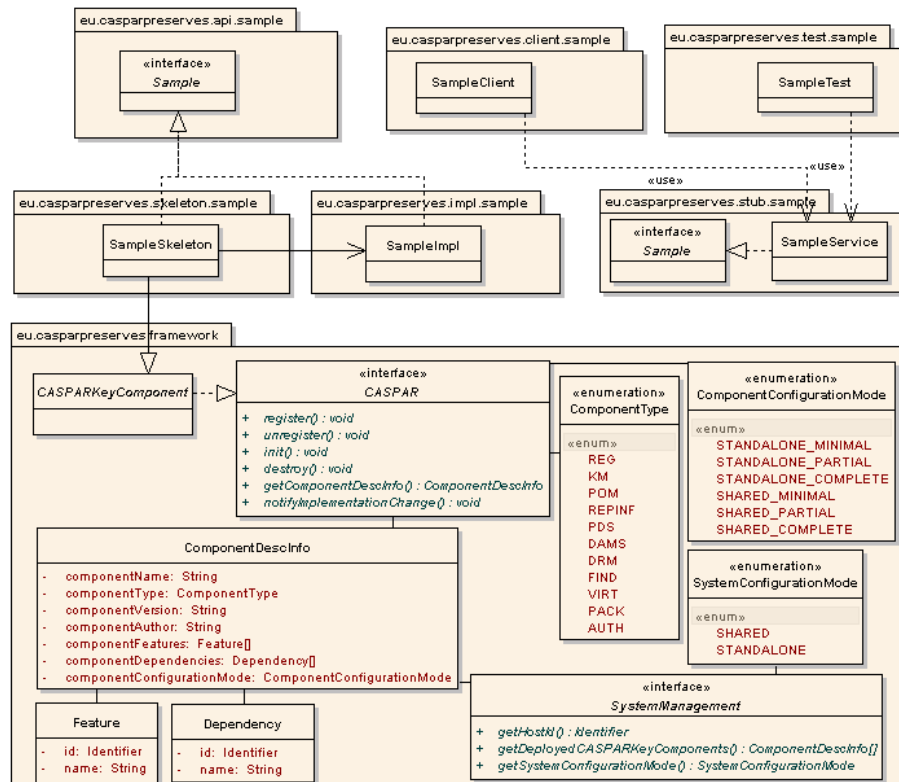


Figure 2 Main CASPAR Packages

## 4.2 GUIDANCE FOR DEVELOPERS

In the following sections the CASPAR Architecture Team shows step by step how to develop a CASPAR Key Component (Sample). You can find the code sample at the CASPAR SVN repository.

### SERVER SIDE

#### 4.2.1 Step 1 – Define the component interface

```
package eu.casparpreserves.api.sample;

public interface Sample {

    /**
     * hello operation
     * @param name of the caller
     * @return welcome to the caller
     */
    public String hello(String name);
}
```

#### 4.2.2 Step 2 – Implement the component interface

```
package eu.casparpreserves.impl.sample;
```





```
import eu.casparpreserves.api.sample;

public class SampleImpl implements Sample {

    public SampleImpl(){

    }

    public String hello(String name) {
        return new String("Welcome to CASPAR " + name);
    }
}
```

#### 4.2.3 Step 3 – Define the component skeleton

```
package eu.casparpreserves.skeleton.sample;

import javax.jws.WebService;
import javax.jws.WebMethod;

import eu.casparpreserves.api.sample.Sample;
import eu.casparpreserves.framework.*;
import eu.casparpreserves.impl.sample.SampleImpl;

@WebService(name="Sample",serviceName="SampleService")
public class SampleSkeleton extends CASPARKeyComponent implements
Sample {

    private final static Sample SAMPLE = new SampleImpl();

    public SampleSkeleton(){
        /*
         * setting component descriptive information
         */
        componentDescInfo = new ComponentDescInfo();
        componentDescInfo.setComponentAuthor("Caspar Architecture
Team");
        componentDescInfo.setComponentName("CASPAR Sample");
        componentDescInfo.setComponentVersion("version 0.0.1");
        componentDescInfo.setComponentType(ComponentType.FIND);
    }

    @WebMethod
    public ComponentDescInfo getComponentDescInfo() {
        return super.getComponentDescInfo();
    }

    @WebMethod
    public String hello(String name) {
        return SAMPLE.hello(name);
    }
}
```





#### 4.2.4 Step 4 – Deployment descriptor files

Under `sample/etc` folder there are three files in order to describe how the component can be deployed as a JAX-WS Web Service:

- **deploy-targets.xml** – define ant task in order to deploy on a proper application container;
- **sun-jaxws.xml** – define the endpoint of the service, its implementation class and its context on application container, according to JAX-WS standard;
- **web.xml** – standard deployment descriptor for web application.

#### 4.2.5 Step 5 – Generation of server artifacts

A **build.xml** is provided in order to build server and client artifacts according to JAX-WS standard.

The **server** ant task can be executed in order to build and deploy server artifacts.



If artifacts are deployed on Apache Tomcat, it's necessary to deploy the JAX-WS libraries (available at **CASPAR** software SVN repository `software/java/framework/jaxws/lib`) on the Tomcat shared libraries folder (i.e. `CATALINA_HOME/shared/lib`).

### CLIENT SIDE

#### 4.2.6 Step 6 – Generation of client artifacts

Client artefacts can be obtained by executing **client-stub** ant task. A set of classes are generated by the tool **wsimport** provided by JAX-WS tools and invoked in `client-stub` ant task. The generated classes are in `eu.casparpreserves.stub.sample` package.

#### 4.2.7 Step 7 – Define client component

In order to facilitate interactions with the set of generated client side classes, a `SampleClient` class is provided. It implements `Sample` interface generated in client artifacts and simply delegates method implementation to `Sample` class by using the generated `SampleService`.

```
package eu.casparpreserves.client.sample;

import java.net.*;
import java.util.*;
import javax.xml.namespace.QName;
import eu.casparpreserves.stub.sample.*;
/*
 * To be compiled and executed only after generating the stub:
 * "ant client-stub" (see build.xml)
 */
public class SampleClient implements Sample{

    private Sample port;

    public SampleClient(URL wsdlLocation){
        QName sQName =
```





```

    new QName("http://sample.skeleton.casparpreserves.eu/",
              "SampleService");
    port = new SampleService(wsdlLocation, sQName).getSamplePort();
}
public ComponentDescInfo getComponentDescInfo() {
    return port.getComponentDescInfo();
}
public String hello(String arg0) {
    return port.hello(arg0);
}
}

```

## ADVANCED CLIENT SOLUTION: SERVICEFACTORY

ServiceFactory is an utility class that creates Service objects, given the service class and the service's URL. That factory allows to simplify client application accessing to JAX-WS Services, and for that reason it has been promoted as a common (framework) utility for all CASPAR Key Components. The code below shows how to use it for invoking a Sample service.

```

package eu.casparpreserves.test.sample;

import java.net.*;
import eu.casparpreserves.client.ServiceFactory;
import eu.casparpreserves.skeleton.sample.*;

public class SampleManagerTest {

    private SampleManager port;

    public SampleManagerTest(URL wsdlLocation) {
        SampleManagerService notMan =
ServiceFactory.createService(SampleManagerService.class,
wsdlLocation);
        port = notMan.getSampleManagerPort();
    }

    public SampleManager getPort() {
        return port;
    }

    /**
     * @param args
     * @throws MalformedURLException
     */
    public static void main(String[] args) throws
MalformedURLException {
        SampleManagerTest s = new SampleManagerTest(new
URL("http://localhost:8080/sample/SampleManager?wsdl"));
        SampleManager sampleManager = s.getPort();
        ComponentDescInfo info =
            sampleManager.getComponentDescInfo();
        System.out.println("Called the component " +

```





```
        info.getComponentName());
    System.out.println("Author of the component " +
        info.getComponentAuthor());
    System.out.println("Version of the component " +
        info.getComponentVersion());
    String welcome = sampleManager.hello("new developer");
    System.out.println(welcome);

}
}
```

### 4.3 NEXT STEPS

It is clear that the *Spring Framework* is a useful and practical solution to apply the Inversion of Control pattern in the **CASPAR** Key Components. And specifically, the concepts of beans and of the dependencies can be applied to manage the configuration of each **CASPAR** Key Component. Further investigations are needed of *Spring* as well as the new technology proposed by Eclipse: *RAP*.

*RAP* will offer the ease of development of Eclipse, Eclipse extensibility and user experience by reusing Eclipse technology for distributed applications and by encapsulating *AJAX* technologies into simple-to-use Java components

Web applications remain the appropriate approach for many usage scenarios, e.g. for applications with zero-install access requirements. With centralized installation and administration, web applications offer low administration and production costs. Furthermore, they can provide high scalability and good performance.

But associated with the classic approach of HTML-based web applications is a higher expenditure of resources during the development and maintenance process due to the technology mix that is involved. With *AJAX* this becomes an even bigger problem.

By encapsulating the *AJAX* technologies in a component library one major road block to a more efficient development model can be eliminated. By offering a plug-in model on the server and providing *Eclipse* workbench UI capabilities developers can leverage their experience with *Rich Client Applications* and profit from the proven *Eclipse* development model to create Rich Internet Applications in a streamlined fashion.





## 5 FURTHER INVESTIGATIONS

In the preceding sections it has been necessary to relate the framework to specific technologies: Web Services, the JAVA language, the Spring framework. The framework has to be implementable with current technologies, and yet as discussed must be independent of those technologies. The CASPAR project is already in a position to identify and evaluate some other approaches that may offer alternatives for implementation, or allow the extension of the framework in new directions (for example, resource discovery).

### 5.1 DATA EXCHANGE MECHANISM

#### 5.1.1 Motivation

Exchange of data is obviously an essential part of a preservation system built with CASPAR components—for example, in passing SIPs and DIPs. Exchange of data between components can be implemented in different ways; in general, performance issues are not critical in a long-term preservation perspective.

#### 5.1.2 Issues and approaches

An exchange of data takes place when some data is taken from a source “structured following the source\_schema” and transformed into data “structured following the destination\_schema”, so that the target data is an accurate transformation that represents the source data.

There may be several ways to make the transformation, in which case we must identify and justify a “best” choice of solutions. Just looking at CASPAR system, it seems reasonable to use SOA (Service Oriented Architecture).

#### 5.1.3 Technologies

In this approach Spring and Mule could help <<services communication>> in a Java-based SOA.

### 5.2 MESSAGING MECHANISMS

#### 5.2.1 Motivation

Message passing is key to the interaction of the CASPAR components, seen as services. Some issues arise that are not found in more ‘normal’ software developments: particularly that messages which are exchanged may not be acted upon until considerable time later, when components may have evolved.

#### 5.2.2 Issues and approaches

It is a general principle that messaging should not influence either the “business interface” or the “core implementation”. The CASPAR system should be built in such a way that the Business Interface and Core Implementation are independent of the messaging, however at various points CASPAR has to deal with sending, routing, and receiving of messages.

In general terms the interchange may be described as follows. The Message is sent by using the “client”. The client is a facade that implements the service interface and delegates implementation to the remote server/service provider. This facade is put into the calling component in the classic manner, without modifying the original logic-code.

The Message is received by the server, which unpacks it and calls an embedded (injected) service implementation.

It can then return the result back to the client or modify a common white-board/registry. The service provider implements the ‘communications interface’. This interface is exposed by the service provider.





The "message policy" separate configurations. The client is a facade that implements the service's business interface, in-packs arguments into messages, dispatches them to the message broker solution, then receives and un-packs responses.

The service provider performs the inverse function receiving messages un-packs them, calling an embedded implementation, receiving, in-packing and returning the results.

We can then inject, into any user of a service, either the direct business implementation, or the communications client. In this way we can choose, at deployment time, whether or not the two services are separated by messaging.

The client accepts a simple interface (injected at run-time) for sending a serializable Java object. The server is a simple class that accepts and returns a serializable object; the main processing is handled by an embedded service implementation which, again, is injected at run-time. Both servers and clients have a simple, regular structure that allows easy generalisation of related functionality, like caching. It could be a great solution the implementation of a <<channel abstraction>>, done for example by the usage of **Mule** that supports many messaging protocols and can itself be extended further.

Common patterns in messaging can be abstracted to separate libraries. This should be restricted to have no dependencies on other packages (except libraries).

Services interfaces could be divided in 2 classes: a first one is that we have called the business and a second one is the communication interface. In our case we are most concerned with relatively closed sub-systems where cooperating services are written in Java. In this case the business interface is the most important. But the communication interface is important as it describes logical components of the system which become more important as the system matures, acquiring connections with external processes.

A Delay on the implementation choice allows services to be "collected" for testing even if they are deployed separately: instead of injecting the communications client, the implementation is used directly. This allows 'integration tests' to be run on developer's machines, without the need for complex deployment.

### 5.2.3 Technologies

It is worth noting at this point some tools which may be useful in this development.

Spring is a "framework". Here we could restrict the examination of it only to an analysis concerning the inversion of control. The Spring container allows one to build applications using/assembling the classes in the implementation package. It eliminates the need for factories and singletons, and leaves the definition of "how instances are constructed" to run-time. This behaviour goes well with the "interface based" approach described.

Mule can be considered as the "Spring messaging tool", that connects "bean/components" with messaging-coupling technologies. Mule creates Java objects from a Spring description and, at the message reception, it calls an instance method, passing the message as an argument. Similarly, it can take the value returned by the method and treat it as a response method.

<http://mule.mulesource.org/display/MULE/Spring>

## 5.3 COMMUNICATION IN A DISTRIBUTED ENVIRONMENT

### 5.3.1 Motivation

In order to allow maximum flexibility and preservability, the CASPAR framework is conceived as a distributed system. That is to say, the components may run and offer their services on distributed machines.

In a distributed system a fundamental aspect is the inter-relation and inter-communication between different components that live in remote hosts. They have to exchange data and messages in order to



perform the task, as in an embedded system, but here the distributed nature of the system has several new aspects. It is therefore appropriate to review the approaches that are available.

### 5.3.2 Issues and approaches

In a distributed environment a component running on a remote machine (or also running on the same machine but in a different environment) could be accessed by find/locate. So the principal need for these components/services on the distributed system is a well defined identification for services that have to be located.

The way CASPAR could be assembled is shown in the following diagram.

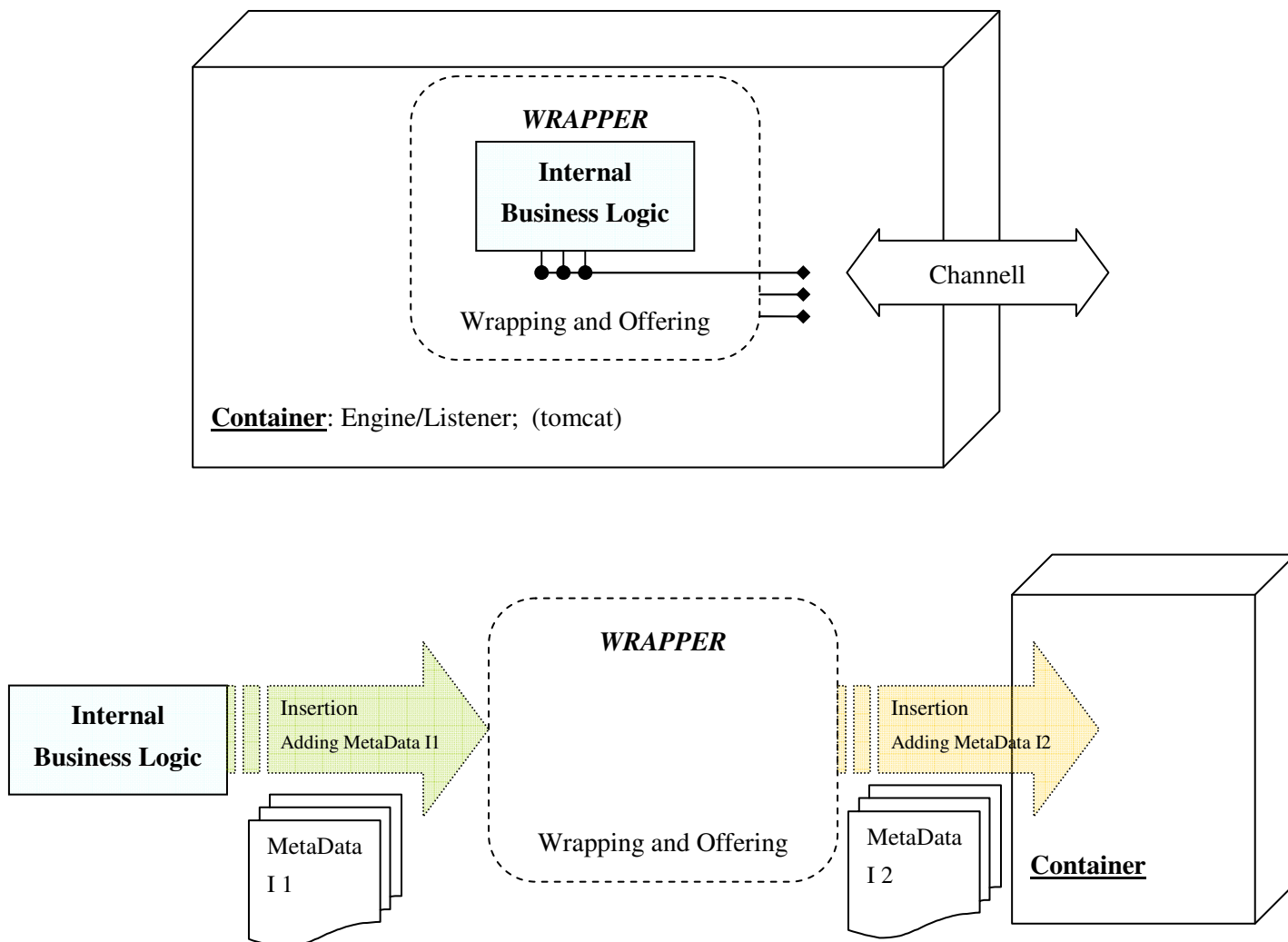


Figure 3 Wrapper

A further issue is search, discovery and redirect by means of brokers. In an ambedded system there are no particular strategies to be adopted: simply the id of the component, managed by container, that drives and redirects communications. If the system is distributed then the container does (substantially) the same things: routes the messages to the endpoint, and these are in the “meta-data” interface declaration for the component.



### 5.3.3 Technologies

Several alternatives to resolve this issue. It depends substantially on the performance requested from CASPAR system. The choice could be addressed by using some available tools or maybe by using only policies or reimplementing these from the beginning.

As tools:

- JXTA
- JINI
- Globus ToolKit
- Condor
- .....

As technology:

- Corba
- UDDI
- P2P
- .....

For the time being we could rely on the assumption that there is no need to use a dynamic distributed system. The end-point URLs of the services are simply exposed in the property files as well known and defined attributes.

If indeed the CASPAR system is composed by components that act as Peer-2-Peer components then a different structure is appropriate: JINI or some catalog for resources (maybe a federated catalog) on which the different services could sign their availability [with some other useful parameters] and to which a component could ask for the most appropriate “service” when it has to fulfill some tasks.

When the system has to deal with a major performance/elasticity then it will be reasonable to adopt some useful technologies (GRID techniques are the result of a leader activity in the Distributed System Study).

## 5.4 REMOTE COMMUNICATION IN SECURE/INSECURE CHANNELS;

### 5.4.1 Motivation

If the CASPAR system is well protected and embedded in a private LAN then it is unnecessary to provide the configuration of secure channels for encrypting the data/messages that flow across the net and for the Authentication/Identification of acting components.

If the CASPAR system might share its domain with the internet (or a public LAN) then the security of channels and Authentication/Identification of acting components is required.

### 5.4.2 Issues and approaches

A centralized Certification Authority could be used that creates a single public/private key pair, used by all the components to:

- Encrypt the communication;
- Identify the actor;

## 5.5 FROM AN EMBEDDED TO A DISTRIBUTED SYSTEM





### 5.5.1 Motivation

The CASPAR project intends to demonstrate that OAIS is a good paradigm for preservation. One aspect of this is distribution of the components, so as to reduce dependency on a central system and to encourage evolution of components independently. To accomplish that CASPAR makes use of several technologies currently available.

In principle CASPAR is a component based system, where each component operates in synergy with the others to fulfil the different “Preservation” operations. Each component is built independently: it has no knowledge of the fact it can run on a particular machine, and the components do not know how and where to act in order to communicate with the external environment. They are aware only of “the CASPAR system”.

### 5.5.2 Issues and approaches

A progressive approach is an optimal choice for CASPAR. In a first iteration a raw model for the CASPAR environment and behaviour will be implemented.

In the first phase each component is installed on a specified engine (such as a Tomcat engine on a networked computer) and can communicate with “partner components” which it needs, starting a call that is appropriately redirected to that component.

The re-direction job will be done (as the beginning) in a partly predefined procedure. Each Point of Contact will be pre-defined and listed in a catalog, and the job of re-direction (done by a Broker) is simply related to that catalog. The broker entity could be either an engine external to all the components, and to which each component refers; or it could simply be embedded in each single component and locally deployed with it.

There will be available a subset of all the components that constitute CASPAR as a whole. They could be hosted in a simple environment : just a set of computers linked in a private LAN. The components are chosen in order to build up a simple trial version of the preservation, in conjunction with the usage of some other tools already available and usable (as described in the D4102 testbed report); considering that:

- 1 we are not confident that all the components could be really available when the integration is done, so the aim is the possibility to build up the system piece by piece, component after component, when it is available;
- 2 starting from an embedded deployment through an iterative process of distribution.

The first point means that each component in the trial-system could be based on a stub that simulates the component but does not offer the complete functionality, just to fill the holes for the whole CASPAR system. The initial end-to-end process has to manage only a restricted set of services that are available and operating.

The end of this iterative process will lead CASPAR to a consistent distributed system in which several CASPAR Components interoperate in order to fulfill the “preservation” and in order to accomplish several non-functional aspects.

The subsequent phase consists in the enlargement of the brokering system.

The components interact with a Broker entity that operates as a work-flow engine: it is implemented as a stand-alone service to which the components send their requests. The brokering is based on a catalog of available components and has to manage the collaboration between them.

The catalog collects many items:

- Component,
- Networks,
- DataObjects,
- Storages,





- Protocols,
- Languages,
- ... and so on.....

The preliminary model is only for the catalog of “available components”, regardless of the possibility that a component migrates to different hosts or maybe is unreachable or is simply present with a reduced set of functionalities or maybe there are several copies of the same components each one offering a different set of functionalities.

The next iteration of the “Integration” will also take into account also the problems related to security. Services, requests, actions... are to be monitored by an appropriate policy of access restriction. In this phase all the channels are encrypted by using a unique CASPAR Key, and each component is identified as a generic “CASPAR Component” that has potentially all the permissions. It is then necessary that each action could be related to the “initiator” (it could be a human user or a component or something else) of the same action in order to establish if there are potential dangerous/unallowed operations.

The Broker thus has to collaborate with the DAMS component and with the Registry/Catalog.

A further issue is component plug-in. Each Key Component must interact with remote Key Components deployed as Services. We are using the word ‘remote’ even if it is possible that several Key Components are hosted on the same machine because we are supposing that each BEAN has a closed visibility and is reachable only through its exposed Business-interface.

So each Key Component must be capable of calling (a priori) remote services/methods and offer its own functionality.

It is composed essentially by:

- “a client”. A sort of engine capable of forwarding on the “network” the requests. In a first phase directly calling the requested service, in the future maybe basing on some BrokeringService.
- “a service-BOX”. An engine-wrapper that acts as a server, listening on designated channels/ports for incoming requests. This BOX is filtering the requests forwarding them to the specified internal Bean capable of expleting the request with its own BusinessLogic.

They could be easily customized as BEANS via some settings on their xml.

In conclusion, we have now decoupled the components of CASPAR system.

In the BOX then could be assembled several functionalities not directly affecting the core business logic of the Key Caspar Component; also in the client embedded.

It is assumed that each BOX or/and Client is directly specified and defined once the Framework has been fixed.

As an example there could be hosted specified services devoted to Authenticity purpose or to some sort of Registry advisory, Brokering subscription, and so on.



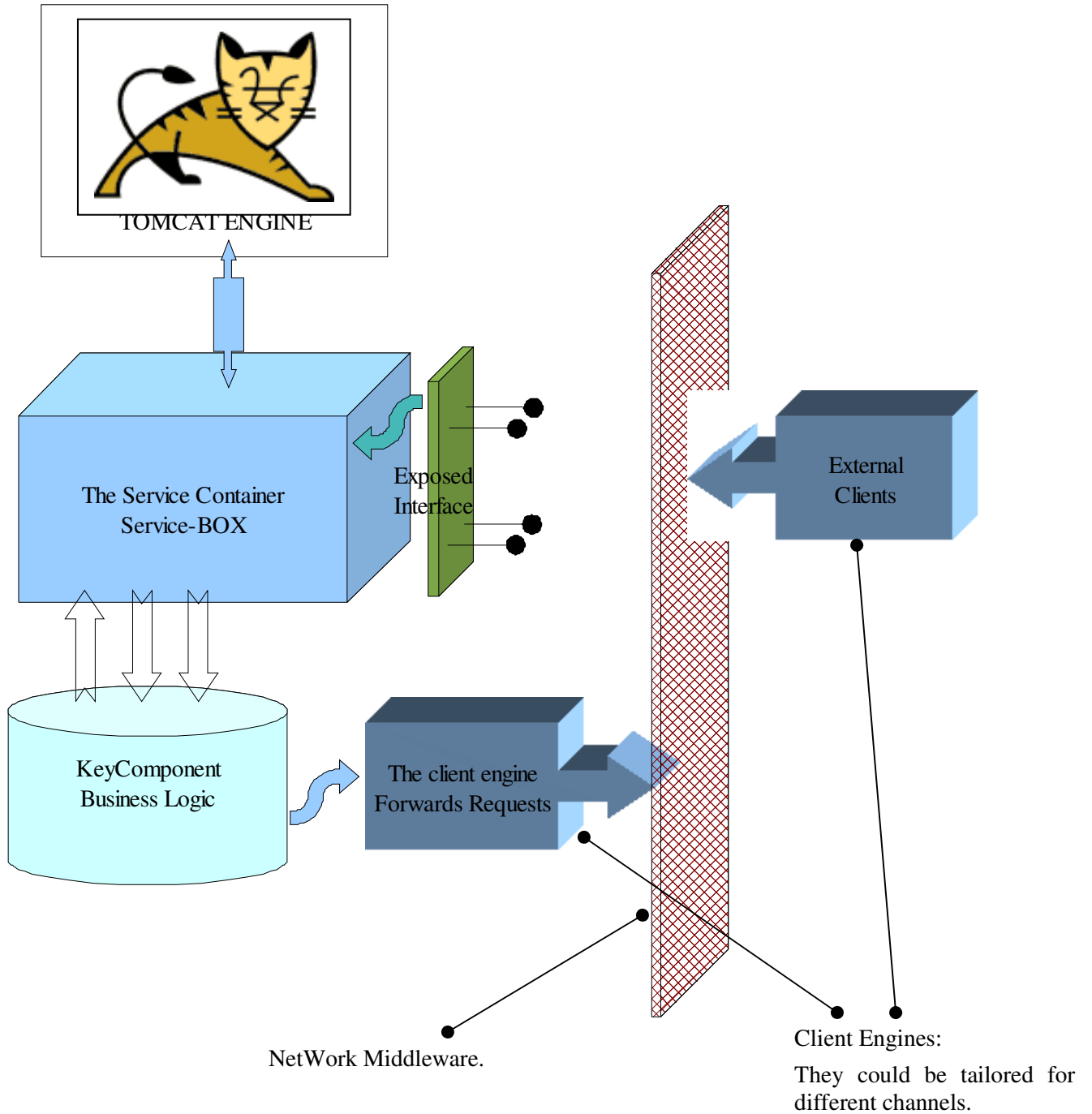


Figure 4: Decoupling the components of the CASPAR system



## 5.6 PROCESS FLOW MANAGEMENT

### 5.6.1 Motivation

Preservation is a complex process, involving the execution of several tasks, with different levels of interconnection. The proper level of automation must be decided on the basis of the specific preservation needs and so the sequencing of actions, required by the OAIS model, if supported by workflow management tools could enhance the system performances.

The CASPAR functional model is a kind of workflow, as the digital assets to be preserved pass through the system and are transformed. In particular applications, a more prescriptive process might be appropriate, tailored to the needs of the material itself or the designated community.

### 5.6.2 Issues and approaches

Process Flow Management generally allows the definition of elements that act as work-units, and that consist of **automated** or **manual steps**. The **steps** can be scheduled and then automatically started or interactively launched.

Each “atomic operation” {step} within the task-chain can have one or more predecessors and one or more successors. So we can look at this chain as a complex graph. When a task-chain starts then the first-step is executed. When it completes, the next step/steps is/are flagged to run, and then in a cascade. There is the possibility that a human (or a machine) has the possibility to directly start a work flow at any step. When this is the case, then the step-launch in the middle chain bypasses all previous steps in the work flow.

In the general case the complex task-chain is a mixture of human/machine action: each is devoted to the monitoring and/or decision-making based on a certain logic.

When “work” flows, there is an activity logging needed for the monitoring. The Manager of the process needs to know when the activities start and when end, if they proceed correctly or there are some inconsistencies/errors.

Preservation entails a complex chain of intermediate steps. Each step has one or more predecessors and one or more successors. To enable the correct flow of jobs in this chain there are 2 main classes of “evolution”:

- A. The centralized managed evolution;
- B. The peer to peer evolution.

The case A refers to the creation of an entity that manages the several jobs during the Process Flow. It monitors the various steps and monitors those that end or eventually reach some defined state.

The case B is related to a different situation, in which each job is treated as a living entity that knows the constitution of the entire chain, or almost knows who is its predecessor and its successor, knows what are pre-conditions and targets. In this peer-to-peer aspect each step directly is responsible to call the start-point for its subsequent step.

The logic of the Management system is broken into simpler logics assigned to each one of the internal jobs of the chain.

In the CASPAR system both the alternatives are possible: a centralized orchestrator and a sequence of internal steps directly invoked by CASPAR components.

### 5.6.3 Technologies

The **Business Process Execution Language (BPEL)** is a language for modelling business processes, and is formatted in an XML way. The concepts of “programming\_in\_the\_large” and “programming\_in\_the\_small” distinguish between two aspects of writing the type of long-running asynchronous processes that one typically sees in business processes.





The “Programming\_in\_the\_small” deals with short behaviour, programmed and planned, that many times are executed as atomic transaction and which allows access to local logic and resources, ACID transaction for example (Atomicity, Consistency, Isolation, Durability).

The “Programming\_in\_the\_large” deals with programming code that represents the high-level state transition logic of a system. In this logic then are encoded information that drive the behaviour, such as when to start, when to wait for actions, when to send (and where) messages and data, what to do when some non-ACID transactions fail and so on.

A number of tools are available, and must be compared according to functionality, robustness and suitability for a digital preservation environment.

- **ActiveBPEL**
- **Service Composition & BPEL**
- **eclipse bpel** <http://www.eclipse.org/proposals/bpel-designer>
- **eclipse tptp** <http://www.eclipse.org/tptp/home/downloads/drops/TPTP-4.1.0.html>
- **apache agila** <http://incubator.apache.org/projects/agila/index.html>
- **twister** <http://sourceforge.net/projects/wf-twister/> - <http://wf-twister.sourceforge.net/>
- **Oracle BPEL Designer Eclipse Plug-in**  
<http://www.oracle.com/technology/software/products/ias/bpel/index.html>
- **active bpel** <http://www.activebpel.org/>
- **Apache ODE** <http://ode.apache.org/>
- **Active BPEL Designer Eclipse Plug-in**  
<http://www.active-endpoints.com/products/activebpeldes/index.html>
- **BeXee** - <http://bexee.sourceforge.net/> - BPEL Execution Engine
- **JOpera** - Composition Editor for Eclipse 3.1 - <http://www.iks.ethz.ch/jopera>
- **Eclipse BPEL** - <http://www.eclipse.org/bpel/>

Apache ODE 1.1 has by far has the best performance, followed by ActiveBPEL 4.0. Oracle BPEL Process Manager is the slowest among the three. It may not be a fair comparison, as both Apache ODE and ActiveBPEL are running on Apache Tomcat, whereas Oracle BPEL has its own application server. However, for critical application, especially when high performance is required, and high load is expected the Apache ODE could be better, if BPEL is absolutely needed.

Regarding development environment, Oracle BPEL stands out in this area. Oracle JDeveloper can be used to develop the BPEL flow, which is very user friendly. Note that Oracle has its own extensions to the WS-BPEL standards.

ActiveBPEL 4.0 has a Eclipse-based designer. For Apache ODE, the Intalio Designer, which uses Apache ODE as its underlying BPEL engine.





## 6 DEVELOPMENT SUPPORT TOOLS FOR CASPAR

### 6.1 STANDARDS AND METHODS

#### 6.1.1 OO design and development

The OO paradigm, to be adopted for the system architecture, is a valid choice in order to develop a tool-set scalable and integrable within different environmental components. It is based on several techniques, including inheritance, modularity, polymorphism, and encapsulation.

Object-oriented Programming is based on a collection of “cooperating objects” as opposed to a traditional view in which a program may be seen as a list of instructions to be computed. In OOP each object receives messages, processes data, and sends messages to other objects. So the “Object” can be thought as an independent “machine” with a defined responsibility.

By objectifying software modules OOP promotes flexibility and maintainability for SW systems and programming; and it is useful in large-scale software engineering. OOP is tending today towards Aspect Oriented Programming (AOP) [see **SPRING** below] , which with aspect-oriented software development (AOSD), attempts to aid programmers in the separation of concerns, specifically crosscutting concerns, as something that aids modularization. AOP does so using primarily language changes, while AOSD uses a combination of language, environment, and method.

The Concern Decoupling entails breaking down the program into distinct parts that overlap (as functionality) as little as possible, and this is not a feature of OOP, as all programming methodologies support a separation and encapsulation of concerns into entities: i.e. procedures, packages, classes, and methods... but OOP emphasizes this aspect.

Any AOP language has some crosscutting expressions that encapsulate the concern in one place. The difference between AOP languages lies in the power and safety, but mostly the Usability/Re-Usability of the constructs provided is the milestone. AOP treats “method executions” and “field references” as points of junction. In them the developer can write a point cut in order to match operations on specific fields, and actions to be executed when the field is actually set.

#### 6.1.2 UML

UML is a Modelling Language that for its flexibility and extensibility is the most adaptable to a modern software system description, and its usage combined with Diagrams makes more simple the non-technical approach.

#### 6.1.3 Design

The CASPAR Project needs a shared work-space in order to enable the exchange, among project partners, of information and design choices. The CASPAR project is spread among several countries and organizations, so a collaborative environment could assure synergy and a better coordination between team-components (and this could be stronger and better if realized in an automatic way).

The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software is of course a CASPAR need.

The UML consists of approximately 9 diagram types in v1.5 and 11 in v2.0 which between them allow the visual representation of both the static relationships and the dynamic behaviour of a system. The reason for the approximation is that diagrams like Object and Package are not always considered to be diagrams in their own right.

Enterprise Architect (EA) supports the UML 2.0 extensions which include the two new diagram types, Interaction Overview and Timing and also the extended functionality like fragments and composite structure. Objects can be drawn on any of the other diagrams.





EA provide very powerful support for UML design and development, is extensible to provide extra functionality. EA offers excellent language support and very impressive UML 2.0 compatibility and

If the latest UML version is a priority, then EA is an obvious choice. It is shipped with very effective version control which even supports baseline fixing and allows the tracing and management of requirements on diagram elements, it offers excellent support for the full project lifecycle and good round trip engineering without the need for additional products.

#### 6.1.4 Development

OASIS, depicts only the major blocks of a logical construct. In the OASIS perspectives they can find place many different implementations because the directives proposed by the OASIS aren't very prescriptive:

An OASIS (Open Archival Information System) is an archive, consisting of an organization of people and systems, that has accepted the responsibility to preserve information and make it available for a Designated Community, maintaining in a "long term preservation", even if the OASIS itself is not permanent. "Long term" is long enough to be concerned with the impacts of changing technologies, including support for new media and data formats, or with a changing user community.

Under the CASPAR ideas we assume the presence of some logical entities, the way they operate are only showed as general aspects. The cooperation and detail definition is completely free.

The assumption is that CASPAR should be built following a P2P coupling between key components is the leading aspect: it will be possible to modify the way CASPAR is assembled... the Orchestrator then is a sort of message-dispatcher.

The communication among the key components is a focal point of investigation.

It could be thought as a RMI protocol, so the web-service paradigm could be an efficient way to achieve the goal. If we consider that different components could run either on the same machine or on remote machines then the intercommunication mechanism as the RMI is the most adequate. To fulfil an RMI communication we can adopt the web service soap protocol, but also different methods could be adopted. A point of investigation is in fact the possible stratification/decoupling of CASPAR components from underlying communication services.

Also the overall decoupling of internal services (functionalities offered by CASPAR components) is an important factor: this approach makes the system more adaptable and scalable.

#### 6.1.5 Configuration Control

Strong needs for configuration control procedures and tools: the project is complex, modular and involves different actors distributed in various physical locations

Configuration control has these meanings:

1. Management of security features and assurances through control of changes made to hardware, software, firmware, documentation, test, test fixtures and test documentation of CASPAR system, throughout the development and operational life of a system. Main point is Source Code Management or/and revision control.
2. Control of changes, made to the hardware, software, firmware, and documentation.
3. Control and adaptation during the evolution of CASPAR project : Software configuration management.
  - 3.1. the storage of the entities produced during the software development project, sometimes referred to as component repository management.
  - 3.2. the activities performed for the production and/or change of these entities (engineering support).





4. Once established a configuration, the evaluating and approving changes to the configuration and to the interrelationships among system components/ system blocks.

We focus the attention on the: Software CM (SCM) and Operational CM

- SCM focuses on managing source code during development,
- Operational CM focuses on managing the configuration items (such as Software, Hardware, documentation, and service definitions) within the infrastructure.

The version control (management of multiple revisions of the same unit) has the responsibility of taking changes to documents/codes/electronic-resources, these changes are identified by incrementing an associated number or letter code, termed the "revision" (chronologically associated with whom makes the change).

SVN allows multiple developers to be editing the same file at the same time, and offers facilities to merge changes into the central repository, so the improvements from the first developer are preserved when the other programmers check in. The concept of a reserved edit can provide an optional means to explicitly lock a file for exclusive write access, even though a merging capability exists.

#### 6.1.6 Development and Configuration Control Tools and Frameworks

- **SVN** <http://subversion.tigris.org/> (see 6.1.6.1)
- **GForge** <http://gforge.org/> (see **Error! Reference source not found.**)
- **Ant** <http://ant.apache.org/> (see 6.1.6.3)
- **CruiseControl** <http://cruisecontrol.sourceforge.net/> (see 6.1.6.4)
- **Maven** [maven.apache.org](http://maven.apache.org/) (see 6.1.6.5)
- **Spring** <http://www.springframework.org/> (see 6.1.6.6)
- **Mule** <http://mule.mulesource.org/> (see 6.1.6.10)
- **Eclipse** [http://www.eclipse.org](http://www.eclipse.org/) - IDE

##### 6.1.6.1 SVN

Subversion (SVN) is an open source version control system. It allows users to keep track of changes made over time to any type of electronic data. Typical uses are versioning source code, web pages or design documents. It has the following features.

- Commits are truly atomic operations. Interrupted commit operations do not cause repository inconsistency or corruption.
- Renamed/copied/moved/removed files retain full revision history.
- Directories, renames, and file metadata are versioned. Entire directory trees can be moved around and/or copied very quickly, and retain full revision history.
- Versioning of symbolic links.
- Native support for binary files, with space-efficient binary-diff storage.
- Apache HTTP server as network server, WebDAV/DeltaV for protocol. There is also an independent server process that uses a custom protocol over TCP/IP.
- Branching and tagging are cheap (constant time) operations.
- Natively client/server, layered library design.
- Client/server protocol sends diffs in both directions.





- Costs are proportional to change size, not data size.
- Parsable output, including XML log output.
- Open Source licensed — "CollabNet/Tigris.org Apache-style license"
- Internationalised program messages.
- File locking for unmergeable files ("reserved checkouts").
- Path-based authorization for svnserve.
- Python, Ruby, Perl, and Java language bindings.
- Full MIME support - the MIME Type of each file can be viewed or changed, with the software knowing which MIME types can have their differences from previous versions shown.

### 6.1.6.2 GFORGE

GForge was developed by the Open Source community as an environment in which to host projects in a way that the code, documentation, binaries etc. were publicly accessible to all who wished to see them, and members of the public could use the software that was developed, and contribute feedback, bugs, ideas and suggestions, and even help to develop code/modules/documentation/resources for the software.

Traditionally it was used for software projects, although there is really no reason why it cannot be used to develop hardware or silicon projects also.

Generally, everyone needs to have read access to the data associated with a project, with (some of) the developers having write access to the data. Usually there is a maintainer of the code (the project leader or the person who registered the project) and contributors who email any changes to the code that they developed - bug fixes, additional functionality - which the maintainer adds to the code in the CVS tree upon verification that it was correct/clean/maintainable/useful.

GForge can provide a centralized access point for several useful utilities and tools which could be used in a project. Some of these tools include:

1. A version control repository (SVN)
2. Mailing lists
3. Discussion forums
4. Bug tracking
5. A web interface to SVN
6. Task lists
7. A website which provides some usage statistics, including the project members, the number of mailing lists, SVN statistics, the number of items in the discussion forums, etc.

### 6.1.6.3 ANT

Apache Ant is a software tool for automating software build processes. It is similar to make but is written in the Java language, requires the Java platform, and is best suited to building Java projects.

The most immediately noticeable difference between Ant and make is that Ant uses XML to describe the build process and its dependencies, whereas make has its Makefile format. By default the XML file is named build.xml.





#### **6.1.6.4 CruiseControl**

CruiseControl is a Java-based framework for a continuous build process. It includes, but is not limited to, plug-ins for email notification, Ant, and various source control tools. A web interface is provided to view the details of the current and previous builds. It allows one to perform a continuous integration of any software development process.

CruiseControl is composed of 3 main modules:

- the build loop: core of the system, it triggers build cycles then notifies various listeners (users) using various publishing techniques. The trigger can be internal (scheduled or upon changes in a SCM) or external. It is configured in a xml file which maps the build cycles to certain tasks, thanks to a system of plug-ins. Depending on configuration, it may produce build artefacts;
- the legacy reporting allows the users to browse the results of the builds and access the artefacts;
- the dashboard provides a visual representation of all project build statuses.

This modularity allows users to install CruiseControl where it will best fit their needs and environment.

#### **6.1.6.5 MAVEN**

Maven is a software tool for Java programming language project management and automated software build. It is similar in functionality to the Apache Ant tool (and to a lesser extent, PHP's PEAR and Perl's CPAN), but has a simpler build configuration model, based on an XML format. Maven is hosted by the Apache Software Foundation, where it was formerly part of the Jakarta Project.

Maven uses a construct known as a Project Object Model (POM) to describe the software project being built, its dependencies on other external modules and components, and the build order. It comes with pre-defined targets for performing certain well defined tasks such as compilation of code and its packaging.

A key feature of Maven is that it is network-ready. The core engine can dynamically download plug-ins from a repository, the same repository that provides access to many versions of different Open Source Java projects, from Apache and other organisations and developers. This repository and its reorganized successor, the Maven 2 repository, strives to be the de facto distribution mechanism for Java applications, but its adoption has been slow. Maven provides built in support not just for retrieving files from this repository, but to upload artefacts at the end of the build. A local cache of downloaded artefacts acts as the primary means of synchronizing the output of projects on a local system.

#### **6.1.6.6 SPRING**

Spring Framework does not enforce any specific programming model, but it can be considered as an alternative and replacement for the Enterprise JavaBean model: JavaBeans-based configuration management, applying Inversion-of-Control principles, specifically using the Dependency Injection technique. This aims to reduce dependencies of components on specific implementations of other components.

Spring can be considered as a collection of smaller frameworks. Most of these frameworks are designed to work independently of each other yet provide better functionalities when used together. These frameworks are divided along the building blocks of typical complex applications:

- Inversion of Control container;
- Aspect-oriented programming framework;
- Data access framework: solutions to technical challenges that are reusable in a multitude of Java-based environments (relational database management systems on the Java platform using JDBC and Object-relational mapping tools);





- Transaction management framework: harmonization of various transaction management API's and configurative transaction management orchestration for Java objects;
- Model-view-controller framework: HTTP and Servlet based framework;
- Remote Access framework: configurative RPC-style export and import of Java objects over computer networks supporting HTTP-based protocols, RMI, CORBA and web services (SOAP);
- Authentication and authorization framework: configurative orchestration of authentication and authorization processes;
- Remote Management framework: configurative exposure and management of Java objects for local or remote configuration via JMX;
- Messaging framework: configurative registration of message listener objects for transparent message consumption from message queues via JMS, improvement of message sending over standard JMS API's;
- Testing framework: support classes for writing unit tests and integration tests;

Central in the Spring Framework is its Inversion of Control container that provides a consistent means of configuring and managing Java objects. This container is also known as BeanFactory, ApplicationContext or Core container.

Examples are: creating objects, configuring objects, calling initialization methods and passing objects to registered call-back objects. Many of the functionalities of the container together form the object lifecycle which is one of the most important features it provides.

Objects that are created by the container are also called Managed Objects or Beans. Typically the container is configured by loading XML files that contain Bean definitions. These provide all information that is required to create objects. Once objects are created and configured without raising error conditions they become available for usage. Objects can be obtained by means of Dependency lookup or Dependency injection.

<<Dependency lookup>> is a pattern where a caller asks the container object for an object with a specific name or of a specific type.

<<Dependency injection>> is a pattern where the container passes objects by name to other objects, either via constructors, properties, or factory methods.

The Spring Framework has its own Aspect-Oriented-Programming framework that is interception-based and configured at runtime. Spring provides also features for the data access framework.

- Resource management;
- Exception handling;
- Transaction participation;
- Resource unwrapping;
- Abstraction for BLOB and CLOB handling;

All these features become available when using Template classes provided by Spring for each supported framework. Moreover the Spring's transaction management framework brings an abstraction mechanism to the Java platform, with it Spring ships a PlatformTransactionManager for a number of transaction management strategies.

Because Spring MVC uses the Spring container for configuration and assembly, web-based applications can take full advantage of the Inversion of Control features offered by the container.





Spring's Remote Access framework is an abstraction for working with various RPC-based technologies available on the Java platform both for client connectivity and exporting objects on servers. The most important feature offered by this framework is to ease configuration and usage of these technologies as much as possible by combining Inversion of Control and AOP.

#### 6.1.6.7 Spring-ws

Spring Web Services (Spring-WS) is in the Spring community and is targeted at the creation of document-driven Web services. Spring-ws facilitates SOAP contract-first service development, and allows the creation of web services using one of the many ways to manipulate XML payloads. It is based on Spring, so it uses the Spring concepts.

Key features of Spring Web services :

- **Mappings** Incoming XML requests are redistributable to any object, depending on message payload, SOAP Action header, or an XPath expression.
- **XML API** Incoming XML messages can be handled not only with standard JAXP APIs such as DOM, SAX, and StAX, but also JDOM, dom4j, XOM, or even marshalling technologies.
- **XML Marshalling** The Object/XML Mapping module in the Spring Web Services distribution supports JAXB 1 and 2, Castor, XMLBeans, JiBX, and XStream. It lives as a separate module, so it's re-usable.
- **Supports WS-Security** WS-Security allows the SOAP messages signature, encryption and decryption, authentication also against them.

#### 6.1.6.8 Java API for xml web-services: JAX-WS

The JAX-WS is an API set (based on Java) for creating web services. It is part of the Java EE platform from Sun Microsystems. It uses annotations, introduced in Java SE 5, to simplify the development and deployment of web service clients and endpoints, as spring-ws also does.

The JAX-WS project is an open source one and lives as a part of the GlassFish, an open source Java EE application server (a branch of the well known Tomcat).

#### 6.1.6.9 AXIS

Axis (from Apache Axis) is an open source (XML based) Web service framework. It is the implementation of the SOAP server using the Java technology.

Subsequent to the Axis there is the Axis2 architecture: an implementation of the Apache Axis2 Web services engine. Axis2 came from the Axis 1.x series project, and both live under the APACHE Software foundation.





### 6.1.6.10 Mule

Mule is an integration framework based on messaging platform that could be used as an Enterprise Service Bus (ESB), handling services and applications using different transport and messaging technologies. Its architecture is scalable with a “broker” that can handle interactions.

Features of Mule:

- Pluggable Connectivity, out of the box support for JMS, JDBC, TCP, UDP, multicast, http, servlet, SMTP, POP3, file, XMPP;
- JBI Integration;
- Flexibility in deployment topologies;
- WebSevices support using XFire, Axis and Glue;
- Spring Integration;

## 6.1.7 Tool collaboration and integration

### 6.1.7.1 GForge & SVN

GForge is a Web-based collaborative development environment. It's based on a fork of the 2.61 SourceForge code, which used to be available via anonymous CVS from VA Software, but has been extensively rewritten and enhanced.

The Gforge server manages projects/packs submissions.

The access to it could be done through a web browser.

GForge registers users and sends notifications, manages accounts.... SVN helps GForge

SVN is used for the check-in check-out, version control system of the project code.

Once a project/pack is created, it can have a Subversion repository on the server. It is important that the code be easy to read and understand. (coding conventions)

### 6.1.7.2 ANT & Maven

ANT is released in a stand-alone distribution, JAVA based, and serves to build the code sources; in conjunction with MAVEN plug-in.

We can use MAVEN either

- with all its functionalities : a compiler tool that manages to decouple the code developed from the different external libraries version (avoiding so the incapsulation of external libs, only tags, and avoiding the needs of including external package source in the code).
- only as an ANT plug-in that manages the libs importing (from existing local directories, or directly from external-net repositories).

### 6.1.7.3 Spring & Mule

See chapter 5.

## 6.1.8 Test Tools

Different levels of testing are requested in the project:

- Low level unitary tests
- Validation tests





Software testing is the process used to measure the quality of the software produced. Usually "test" is oriented towards correctness, completeness, security.... but often it includes technical requirements as capability, reliability, efficiency, portability, maintainability, compatibility, usability. As a process of tech-analysis (made on behalf of stakeholders) intended to reveal information about the product and the context in which it is intended to operate could be seen as "Functional"; including the process of executing a program or application with the intent of finding errors. The "quality" is not an absolute value; it is always related to someone/something. On the complementary side there is the "Non Functional" investigation of Testing.

A definition of testing is "fundamentally an investigation, about questioning the system in order to evaluate it", where "questions" are operations that tester attempts to execute [with the system/using the system] and the "answers" are the system behaviour as a reaction to that probe. Testing as "inspection" is called "static testing", whereas the testing made through the "run of the system" with a given set of [test cases/environmental conditions] in a given development stage is often referred to as "dynamic testing".

[White/Black] Box test describes which is the point of view of the tester:

- Black-Box testing offers the view of the object-system as an opaque one; inputs data and outputs from the system;
- White-Box testing offers an internal view of the test object and its processes.

Gray-Box test permits to set up or manipulate the testing environment and can view the state of the product after his actions, such as performing SQL queries on a DB, monitoring of http-requests.

#### **6.1.8.1 Verify & Validate**

Verification checks for items, software, conformance and consistency with the specifications. Validation checks if the stuff specified by the user is what the user actually wanted.

#### **6.1.8.2 Test Level**

The "unit-test" is tuned to investigate the minimal software component/module. Each basic component of the system is tested to verify that the design for the unit has been correctly implemented at the detail level.

The "integration-test" examines the existence of defects in the interaction between integrated components/modules: i.e. defects in the interfaces. This kind of test is a recursive one, it starts assuming a certain granularity and progressively augments its scope to larger groups of tested sw components. And that corresponds to an integration of elements of the architectural design that are tested until the system is running as a single system.

The "system-test" examines the system as "integrated" and verifies that meets the requirements:

- Functional ones;
- Non-Functional ones;

The "acceptance-test" is conducted in simulation/effectiveness [alpha-test/beta-test] of the user/client that validates finally the product.

#### **6.1.8.3 Test case & Scenario**

Test-Case collects events, actions, inputs, outputs, expected results, and actual results about a SW system. It is a series of steps (but often steps are contained in a separate test procedure) to which one expected result or expected outcome is attached, it may also contain "prerequisite states" and/or descriptions. The Test-Case has to consider where to store/analyze the actual results. The collection of test cases gives the "test-suite": a test specification. If in this collection we can set also a <<sequence>>, in the sense of a chronological or cause-effect, then it can be called a "Scenario of Test".





#### 6.1.8.4 A possible testing cycle

- a) Test Planning: A Strategy chosen in conformity with the TestBed;
- b) Test Development: Procedures, Scenarios, Test-Cases;
- c) Test Execution;
- d) Test Reporting;
- e) Retesting the Defects ????

I want also notice that among the “errors” there is a part that could be the effect of different errors in configuring the testing in order to match the [devel/production] environment: a set of errors could be elided by using “workarounds” in the production environment.

#### 6.1.8.5 Interfaces and Mock Objects

Often it happens that objects have references to other objects, so testing one object can frequently spill over into testing other objects : classes that interact with a DB, to test them the mistake is to write code that “interacts” with the DB. The unit test must never go out of its own class-scope boundary. The solution is the simulation of the interaction with external body via some interface around it/its clone, and then an implementation of those interfaces with fine tuned “**mock object**”.

#### 6.1.8.6 Techniques & frameworks

A manual approach to unit testing uses a step-by-step instruction. Automation is efficient but if not planned carefully could preclude the achievement of most goals established.

Under the automated approach the effect of isolation is obtained by executing the block within a framework outside of its natural environment, outside of the product or calling context for which it was originally created. During execution of the test cases the framework logs those that fail any criterion, reporting a summary where failed, and depending upon the degree of failure, the framework may halt subsequent testing.

<i>Name</i>	<i>xUnit</i>	<i>Remarks</i>
JTiger		JTiger includes rich assertion abilities, ranging from 'assert true' to 'assert adherence to the Object equals method contract on this type', an Apache Ant task, and reporting capabilities in HTML, XML or plain text format.  <a href="http://www.junit.org/index.htm">http://www.junit.org/index.htm</a>
JUnit	Yes	JUnit is a unit testing framework for Java, in the xUnit family of frameworks, that can be considered the UNIT Test framework.
DBUnit		a JUnit extension to perform unit testing with database-driven programs
JUnitEE		a JUnit extension for testing Java EE applications
GroboUtils		a JUnit extension providing Automated documentation, class hierarchy unit testing, code coverage, and multi-threaded tests.
TestNG		TestNG is designed to cover all categories of tests: unit, functional, end-to-end, integration....
GJtester		GJTester is an implementation of the Computer-Aided Software Testing (CAST) paradigm.  GJTester offers effective error detection mechanism for the full software lifecycle which helps considerably decrease software errors, increases software reliability and quality, and consequently customer satisfaction.





---

## Table 1 –Test Tools

### 6.1.9 Evolution Management

Different versions of OAIS Blocks will be released during the project; and backward compatibility must always be ensured.

We start by using some artefacts: maybe they are blocks from existing systems, Beans available as components, simply to investigate (contextually) the testbed scenario and behaviour. Available components usage is arranged in such a way that a simulation of CASPAR is achieved only throughout a wrapping, or encapsulation of those.

The first release of some system components then could be partially integrated by the wrapped ones.

It is important that the evolution of the project could always be subject to a regression: this in order to verify the variations and/or the improvements.

The history of CASPAR evolution must be preserved as well, then the system has to manage the several version of the system itself (from on side) and the different version that appear during its evolution.

