

INSEMTIVES
FP7-ICT-2007-3
Contract no.: 231181
www.insemtives.eu

INSEMTIVES

Deliverable 3.2.1

Semantic Content Management Platform (Initial Version)

Editor:	Mihail Konstantinov, ONTO
Deliverable nature:	Prototype (P)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M12
Actual delivery date:	M12
Suggested readers:	Developers creating software components of the Insemtives Platform, developers creating case study prototypes, etc.
Version:	1.0
Total number of pages:	18
Keywords:	manual annotation, multi-media annotation, semantic annotation, semantic search, semantic information retrieval

Abstract

The subject of the deliverable is the initial version of the implementation of the Semantic Content Management Platform (initial version). The focus of this document is to describe the current state of the implementation and deliver a guide to the developers who would need to use the covered components.

The initial implementation itself is a simple but fully featured Java service container. There are several base services provided along with the service platform, including a communication framework that enables efficient exchange of data between the services within the same server and across networks.

Disclaimer

This document contains material, which is the copyright of certain INSEMTIVES consortium parties, and may not be reproduced or copied without permission.

All INSEMTIVES consortium parties have agreed to full publication of this document.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the INSEMTIVES consortium as a whole, nor a certain party of the INSEMTIVES consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, and accepts no liability for loss or damage suffered by any person using this information.

[Full project title] INSEMTIVES – Incentives for Semantics

[Short project title] INSEMTIVES

[Number and title of work-package] WP 3 INSEMTIVES platform

[Document title] Semantic Content Management Platform

[Editor: Name, company] Mihail Konstantinov, ONTO

[Work-package leader: Name, company] Borislav Popov, Ontotext AD

[Estimation of PM spent on the Deliverable] 6 PM

Copyright notice

© 2009 Participants in project INSEMTIVES

Executive summary

This document describes the initial version of the Semantic Content Management Platform. The goal of this component in the Insemtives Platform is to be the base of all tools, games, and semantic services that are part of the Insemtives Project. The platform design and features derived from the requirements set in D 3.1 Requirement Analysis and Architectural Design of Semantic Content Management Platform [1].

Based on these requirements, we developed a modular platform for Java services, a powerful structured knowledge database layer and a robust communication framework. The features of the modular platform for Java services are described in detail in Section 4: Basic operations. We believe that the service platform will enable the partners in Insemtives to easily integrate all software deliverables into a single package. We have documented the entire workflow of integrating a sample component in the Insemtives platform.

The robust communication framework is based on the Java Message Service API (JMS). The framework builds on the messaging capabilities of that API and enables the services in the platform to communicate with higher order structures like objects. Although it originated in Java, there are clients for JMS in many software languages. The software clients of the Insemtives platform are thus not limited to Java.

The powerful structured knowledge database layer is based on BigOWLIM - the most scalable and efficient RDF Database. We have combined the capabilities of BigOWLIM with the communications framework to deliver a high performance database - RDFDB - which can be used across the network efficiently.

List of authors

Company	Author
Ontotext AD	Mihail Konstantinov
Ontotext AD	Marin Nozhchev
Ontotext AD	Atanas Ilchev
Ontotext AD	Reneta Popova
Ontotext AD	Gergana Petkova

Table of Contents

Executive summary	3
List of authors.....	4
Table of Contents	5
List of figures	6
Abbreviations	7
1 Communication Framework Overview	8
2 Status.....	10
3 Downloading and Installing	11
4 Basic Operations	12
4.1 Developing a Service	12
4.1.1 Writing the Business Logic.....	12
4.1.2 Annotating the Service.....	13
4.1.3 Deployable Generation	16
4.2 Deploying a Service	16
4.3 Generating a Client	17
References	18

List of figures

Figure 1 Communication Framework..... 8

Abbreviations

JMS Java Message Service

API Application Programming Interface

RDF Resource Description Framework

JAVADOC Java Documentation Generation Tool

1 Communication Framework Overview

The objective of this section is to introduce the overall structure of the INSEMTIVES Platform and the operations it provides.

To facilitate the extensibility and flexibility of the platform we propose highly modular server-client architecture. In order to decouple the platform services as much as possible, we have implemented a communication layer over Java message protocol (*JMS*) [2]. This layer will include Java annotations for services, a framework server for deployment of services and tools for generating Java clients.

Since *JMS* is a well-accepted standard and a lot of implementations are available, it would be easy for the INSEMTIVES toolkit and usecase partners to use the platform, and for the platform implementers to extend it. For example, *ActiveMQ* cross-language clients [3] provide implementations for 16 languages. Also, it supports 6 protocols including *REST* and *WS Notification*.

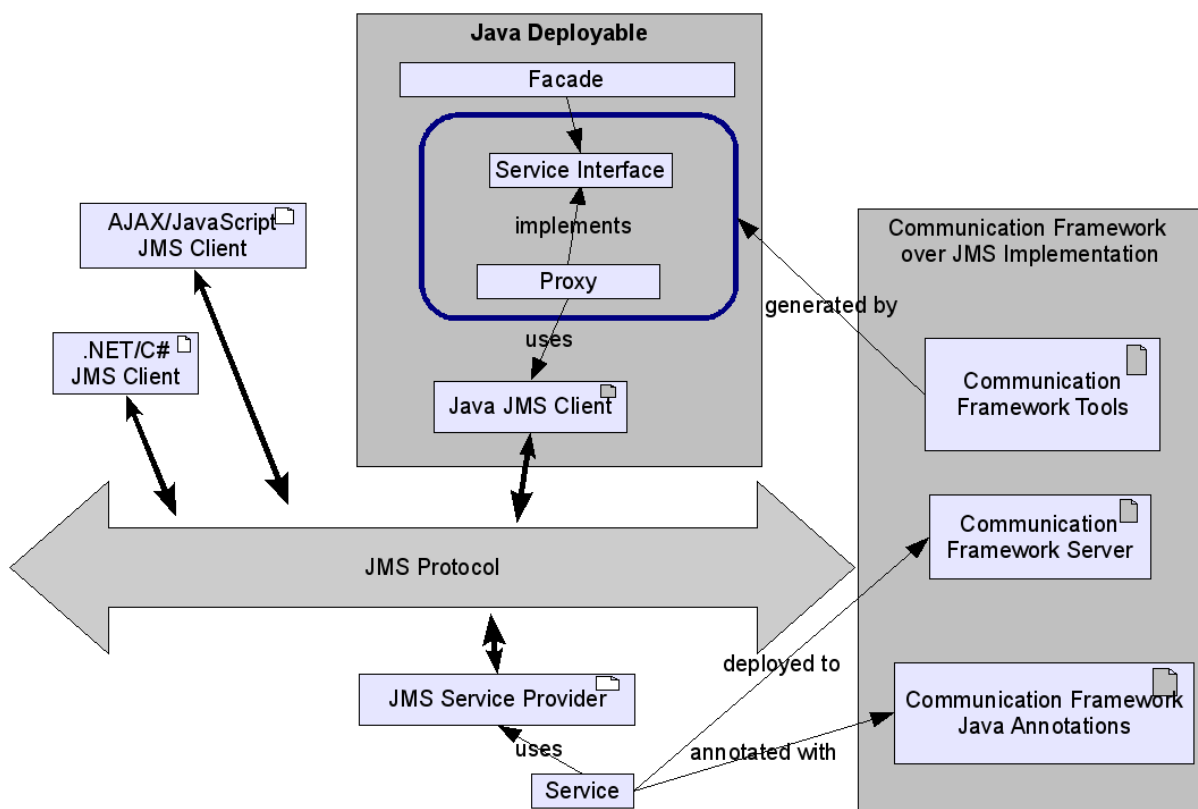


Figure 1 Communication Framework

To implement a **service**, an implementer uses annotations in the package `com.ontotext.platform.server`. These are `Service`, `ServiceMethod`, `Startup`, and `Shutdown`. The first one is used to annotate the class that implements the service with a parameter name - the name of the service. Methods that are going to be exposed to the clients are annotated with `ServiceMethod`. The last two annotations are used when the service needs initialization and cleaning up before finalization.

Once the service is annotated, it is packed and ready for deployment on the communication framework server - the runtime component of the communication framework. We might have one or several communication servers in the deployed infrastructure, depending on the performance requirements. At the client side we have a **proxy** Java class and an **interface**. The former handles all communication with a service transparently, and the latter provides the interface to the service to program against. The proxy and the interface are generated with a command-line tool using the annotated service class.

This type of decoupled communication mechanism addresses the requirements for integration with external components in the three use cases, as well as anticipated toolkit requirements. For the *seekda* usecase, these are requirements R14 to R16 from D3.1.1. [4]: integration with Web API, *seekda* LDAP user authentication, and data exchange with an external database. For the *PGP* usecase, these are FR46 to FR49, FR51 and FR52: user authentication with MTP, integration with TP, and external databases and communication with sandbox server for streaming videos. For the *Telefonica* usecase, these are “Requirement WP5.SEC01. INTEGRATION” - integration with external authentication mechanism, “Requirement WP5.OPE01. PLATFORM ACCESS” - exporting the platform through a lightweight standard-based and protocol-oriented communication mechanism. Additionally, this architecture addresses the general requirement for extensibility since any functionality could be easily integrated as a service.

2 **Status**

At this initial version of INSEMTIVES Platform, we have implemented the *Communication Framework* and the most basic layer above it - namely *RDF-DB*. We have provided documentation describing how to perform the essential operations with the platform. This enables developing other higher-level services, such as *Annotation Service*, that will integrate to the platform and between each other, in highly-decoupled way.

3 Downloading and Installing

A distribution of INSEMTIVES Platform can be obtained from [here](#).

To run INSEMTIVES Platform you need *Java Standard Edition 6* compatible *JVM* and **1GiB** of operative memory. Since INSEMTIVES Platform uses port **61616**, it must not be used by any other applications.

Installation is as simple as de-archiving the distribution in a folder and running appropriate for your OS *platform_start* and *platform_stop* batch or *BASH* scripts. (Note for *ux users might need to run *chmod +x* for all the scripts that need to be executable.)

4 Basic Operations

A minimal complete set of operations that the INSEMTIVES Platform provides is: "Developing a Service", "Deploying a Service" and "Generating a Client".

4.1 Developing a Service

Developing a service constitutes:

- Writing the class that implements the business logic and composes operations as needed, i.e. the **service**;
- Annotating the service so that the **interface**, **proxy**, and **client** could be generated and deployed;
- Generating a deployable artifact (*JAR*) containing the **service**, its **interface** and **proxy**.

4.1.1 Writing the Business Logic

First, we create objects that are going to be sent forth and back with the service.

```
package com.ontotext.platform.qsg;
import java.io.Serializable;
public class Package implements Serializable {
    private static final long serialVersionUID = 2746658752245876605L;

    private String content;

    public Package() {}

    public Package(String content) {
        this.content = content;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }
}
```

All objects used as parameters must implement *java.io.Serializable*.

We continue by developing the service class.

```
package com.ontotext.platform.qsg;
public class PigeonPostalService {
    protected static long sentId;
    protected static long receivedId;

    private Package pack;

    public PigeonPostalService() {}

    public void initialize() {
        sentId=1000;
        receivedId=20000;
    }

    public void shutdown() {
        sentId=0;
        receivedId=0;
    }

    public void send(Package pack) {
        this.pack = pack;
    }

    public Package receive() {
        Package pack = this.pack;
        this.pack = null;
        return pack;
    }
}
```

4.1.2 Annotating the Service

Before annotating you need to configure your build environment.

4.1.2.1 Building with *Maven2*

In your *POM* you have to include the following repository configuration:

```

<repositories>
<!-- Note that the repository URLs should NOT end in / -->
<repository>
  <id>internal</id>
  <name>Archiva Managed Internal Repository</name>
  <url>http://maven.ontotext.com/archiva/repository/internal</url>
  <releases>
    <enabled>true</enabled>
    <updatePolicy>always</updatePolicy>
  </releases>
  <snapshots>
    <enabled>false</enabled>
  </snapshots>
</repository>
<repository>
  <id>bigowl</id>
  <name>Archiva Managed Bigowl Repository</name>
  <url>http://maven.ontotext.com/archiva/repository/bigowl</url>
  <releases>
    <enabled>true</enabled>
  </releases>
  <snapshots>
    <enabled>false</enabled>
  </snapshots>
</repository>
<repository>
  <id>snapshots</id>
  <name>Archiva Managed Snapshot Repository</name>
  <url>http://maven.ontotext.com/archiva/repository/snapshots</url>
  <releases>
    <enabled>false</enabled>
  </releases>
  <snapshots>
    <enabled>true</enabled>
    <updatePolicy>never</updatePolicy>
  </snapshots>
</repository>
</repositories>

```

and add the following dependency

```

<dependency>
  <groupId>com.ontotext</groupId>
  <artifactId>platform-server</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>

```

4.1.2.2 Building Using JARs Provided with the Platform

Just add all *JAR* files that are in the directory *lib*, located in the extracted platform distribution, to your classpath. For example, */opt/platform-distribution-1.0-SNAPSHOT-bin/lib*.

Annotation proceeds by annotating the service class and the methods you want to expose in the service.

```
package com.ontotext.platform.qsg;

import com.ontotext.platform.DocAnnotations.Description;
import com.ontotext.platform.DocAnnotations.PName;
import com.ontotext.platform.server.Service;
import com.ontotext.platform.server.ServiceMethod;
import com.ontotext.platform.server.Shutdown;
import com.ontotext.platform.server.Startup;

@Service(name="PigeonPostal") public class PigeonPostalService {
    protected static long sentId;
    protected static long receivedId;

    private Package pack;

    public PigeonPostalService() {}

    @Startup
    public void initialize() {
        sentId=1000;
        receivedId=20000;
    }

    @Shutdown
    public void shutdown() {
        sentId=0;
        receivedId=0;
    }

    @ServiceMethod
    @Description("Sends a package to the postal")
    public void send(@PName("pack") Package pack) {
        this.pack = pack;
    }

    @ServiceMethod
    @Description("Receives a package from the postal")
    public Package receive() {
        Package pack = this.pack;
        this.pack = null;
        return pack;
    }
}
```

The meaning of the annotation is as follows:

- @Service - marks the class as a service. This is the only mandatory annotation. It has to be applied to the class;
- @ServiceMethod - applied on methods that are to be exported as operations of the service (i.e. available for consumption by the clients);
- @Startup - method(s) marked with this attribute will be invoked after the class is instantiated by the components service. There is no guarantee for the order of the invocation, if there are more than one methods marked as @Startup;

- `@Shutdown` - method(s) marked with this attribute will be invoked before the class instance is released by the components service. There is no guarantee for the order of the invocation, if there are more than one methods marked as `@Shutdown`;
- `@Description` - JavaDoc description for the generated source code. It can be applied to both service methods and their arguments;
- `@PName` - sets the name of the arguments in the generated source code.

4.1.3 Deployable Generation

In order to generate an interface and a proxy for the service, one has to invoke `mkproxy` tool located in the `bin` directory of the platform installation. The parameters are as follows:

```
mkproxy {-iface|-proxy|-client} <class-of-the-annotated-
service> <path-to-the-built-classes-with-the-trailing-slash>
```

In our case, using your shell navigate to the `bin` directory and invoke the following commands:

```
mkproxy -iface com.ontotext.platform.qsg.PigeonPostalService
$PATH_TO_EXAMPLES/target/classes/
mkproxy -prxoy com.ontotext.platform.qsg.PigeonPostalService
$PATH_TO_EXAMPLES/target/classes/
```

Both commands dump output to `sysout`. Get the code, clean it as appropriate and put it in your project's sources. Build the project and the deployable JAR is located in your project's target directory.

4.2 Deploying a Service

Deploying a service artifact constitutes:

- Editing INSEMTIVES Platform config file and copying the deployable artifact to an instance of INSEMTIVES Platform;
- Restarting INSEMTIVES Platform.

Create a `services` directory in your `$PLATFORM_HOME` and copy the deployable JAR there. Stop the platform. Edit the platform's configuration located in `$PLATFORM_HOME/config/platform.ttl`. The format is as follows:

```
@prefix kim_entity: <http://ontotext.com/kim/3/entities#>. @prefix
kim_service: <http://ontotext.com/kim/3/services#>. @prefix rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

kim_service:your-service-name-from-annotation rdf:type
kim_entity:service;
  kim_entity:implementation "fully-qualified-service-class-name";
  kim_entity:classpath "path-to-your-deployable-jar";
  kim_entity:arguments "".
```

In our example, the configuration is as follows:

```
@prefix kim_entity: <http://ontotext.com/kim/3/entities#>. @prefix
kim_service: <http://ontotext.com/kim/3/services#>. @prefix rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
```

```
kim_service:PigeonPostal rdf:type kim_entity:service ;
    kim_entity:implementation
"com.ontotext.platform.qsg.PigeonPostalService";
    kim_entity:classpath "$PLATFORM_HOME/services/platform-experiments-
1.0-SNAPSHOT.jar";
    kim_entity:arguments "".
```

After editing the configuration, start the platform and watch the logger's output. The service should be loaded and initialized.

4.3 Generating a Client

Using your shell, navigate to the bin directory and invoke the following commands:

```
mkproxy -proxy com.ontotext.platform.qsg.PigeonPostalService
$PATH_TO_EXAMPLES/target/classes/
```

```
mkproxy -client com.ontotext.platform.qsg.PigeonPostalService
$PATH_TO_EXAMPLES/target/classes/
```

Both commands dump output to sysout. Get the code, clean it as appropriate, and put it in your project's sources. Build the project and the client is located in your project's target directory.

References

- [1] Requirement Analysis and Architectural Design of Semantic Content Management Platform (Del 3.1)
- [2] Java Message Service (JMS) <http://java.sun.com/products/jms/>
- [3] ActiveMQ Cross Language Clients <http://activemq.apache.org/cross-language-clients.html>
- [4] ISEMTIVES Platform Specification (Del. 3.1.1.)