Universidad Rey Juan Carlos

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA INFORMÁTICA

Curso académico 2008-2009

Proyecto de Fin de Carrera

# Development and integration of an awareness applications manager into ASTRA

| | |
|---|---|
| Autor: | David Rozas Domingo |
| Tutoras: | Soto Montalvo (URJC) |
| | Monica Divitini (NTNU) |

*A mis padres.*
*Gracias por educarme para ser la persona*
*que soy, con mis defectos y mis virtudes,*
*con mis aciertos y mis errores. Os quiero.*

# Acknowledgments

I would like to thank Professors Soto Montalvo and Monica Divitini for giving me the opportunity of making this project possible. I wish to thank also all the people from NTNU, CTI and Telenor involved in ASTRA; specially Alfredo, a fantastic computer scientist but even better friend.

# Abstract

ASTRA (Awareness Services and Systems - Towards Theory and Realization) is a project that researches awareness systems and services that are used for social purposes through the creation of awareness applications. The project discussed in this document is part of the work developed for ASTRA, and it aims the creation and integration of a system to manage the mentioned awareness applications, including functionalities for sharing, tagging, locating, appropriating and adapting them, taking into account the concerns about privacy in terms of visibility for all the involved elements (applications, tags, etc.).

The backbone of ASTRA is "ASTRA SOA" (Service Oriented Architecture), the platform where all the necessary services are offered. It is made up of a group of OSGi bundles which are divided into two subsystems from a high level point of view: ASTRA Node and ASTRA Backend, following a Client-Server model. This document details the process carried on to develop a new set of bundles for both subsystems, which offer the previously mentioned functionalities. It also includes an analysis of an users study evaluation performed at NTNU (Trondheim, Norway) in December 2008 with a preliminary version, and the guidelines to perform a new users study scheduled in October 2009, with emphasis on the searching capabilities evaluation.

# Resumen

ASTRA (Awareness Services and Systems - Towards Theory and Realization) es un proyecto que investiga sistemas y servicios “awareness” empleados para propósitos sociales a través de la creación de aplicaciones “awareness”. El proyecto que se detalla en este documento se enmarca dentro del trabajo desarrollado en ASTRA, y tiene como objetivo la creación e integración de un sistema para gestionar dichas aplicaciones, incluyendo funcionalidades para compartirlas, etiquetarlas, localizarlas, recuperarlas y adaptarlas, haciendo hincapié en la privacidad en términos de visibilidad de todos los elementos (aplicaciones, etiquetas, etc.).

La piedra angular de ASTRA es “ASTRA SOA” (Service Oriented Architecture), la plataforma desde la que se ofrecen todos los servicios necesarios. Está compuesta por un grupo de bundles OSGi que, a alto nivel, se dividen en dos subsistemas que siguen un modelo Cliente-Servidor: ASTRA Node y ASTRA Backend. Este documento detalla el proceso llevado a cabo para desarrollar un conjunto de nuevos bundles para ambos subsistemas, que ofrecen las funcionalidades previamente mencionadas. Se incluye igualmente un análisis de un estudio de usuarios para la evaluación de una versión preliminar llevado a cabo en la universidad NTNU (Trondheim, Noruega) en diciembre de 2008, así como una serie de pautas para el próximo estudio de usuarios programado para Octubre de 2009, con especial hincapié en la evaluación de las nuevas funcionalidades relativas a los procesos de búsqueda.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 What is ASTRA?

ASTRA (Awareness Services and Systems - Towards Theory and Realization) [CM07] is a project (logo in Figure 1.1) that researches awareness systems and services that are used for social purposes. Awareness systems are computer-mediated communication systems that help individuals or groups build and maintain a peripheral awareness of each other, allowing them to stay in touch.

Compared to telephony and video conferencing, awareness systems offer low effort and non-intrusive communication throughout the day. The state of the art is limited to non-functional prototypes without corresponding advances in theory development. The assessment project aims to establish whether such systems serve real and not imaginary user needs, through limited implementation and testing. It will test social presence as the appropriate theoretical framework for these systems and gain a first understanding of requirements for middleware and interactive devices to support awareness services. ASTRA is a FET (Future and Emerging Technologies) project under EU's Framework Programme 6 under contract number IST-2001-39270, running in 2006-2009.

The organizations which are part of the ASTRA consortium are:

- `CTI` (Research Academic Computer Technology Institute), brings in experience in Designing Ambient Intelligence Systems (DAISy). The CTI DAISy research team bridging computer science with interaction design will lead the project and support with research in end user programming tools, design framework, ontologies and modules development for the architecture.

Figure 1.1: ASTRA logo

- `TU/e` (Industrial design), shall bring in expertise in interaction design, user interface technology and qualitative research methods for interaction design. `TU/e`, Technology Management, shall contribute expertise in quantitative methods for assessing user experiences, social presence and experimental design.

- `Telenor`, shall bring in expertise on communication services and software architectures.

- `Philips`, shall bring in experience in field studies and the design of user interfaces for entertainment and leisure. Philips will support user tests in realistic conditions in the HomeLab, a simulated home-laboratory, especially built and equipped for large scale field studies.

- `NTNU` (Norges Teknisk-Naturvitenskapelige Universitet), through Monica Divitini, professor of cooperation technologies in the Department of Information and Computer Science. This project has been developed as part of the work of this organization.

## 1.2 ASTRA applications

A key concept in the ASTRA project is an ASTRA application [DC09]. ASTRA applications represent the mean to transform the services provided by the system into awareness applications. There are two different types of applications: “Nimbus” and “Focus” applications. Nimbus applications are used to make available awareness information to other users, while Focus applications are used to decide what information we are interested in and in which way we want to receive that information. Figure 1.2 shows a schema where we can see the relationship between the two types of application. In this case, the user Alice has decided to create an ASTRA Nimbus application to express her feelings ("thinking of you") when she hugs a pillow. Once this application is published to certain community ("family" in Figure 1.2), the rest of the users that belong to that community can subscribe to it and define a way to receive the information by defining a Focus application. In the case of Figure 1.2, Alice's brother has decided to visualize it using a lamp, which will be turned on when Alice's Nimbus application is triggered. Therefore, the system has been able to capture and send this awareness information in a taylorized way.

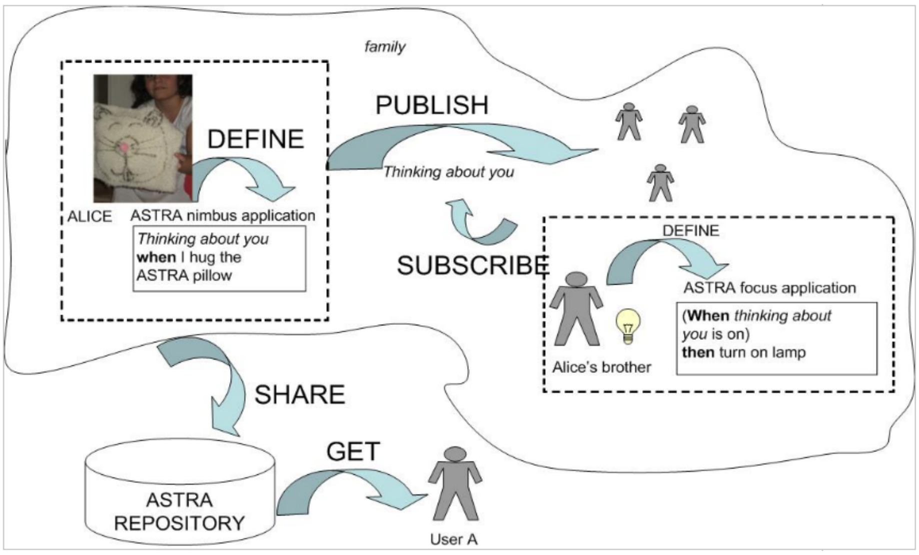


Figure 1.2: ASTRA applications example

## 1.3 Motivations of the project

The aim of this project is to create a system to manage the applications we mentioned in Section 1.2 and integrate it into the ASTRA SOA (see Section 3.2.1.2). Managing comprises fuctionalities for sharing, tagging, locating, appropriating and adapting the applications. Figure 1.3 shows the process of sharing and appropriating applications through a central repository, illustrating the need of customizing the parameters that are going to be shared (I.e.: for privacy reasons) by user A, and the need of adapting the application to its preferences (I.e.: connect it to his physical devices) by user B. This process is potentially complex, therefore functionalities to help the user to perform it are needed, as we will see in Section 4.4.4. It is important to remark the difference between "sharing" and "publishing". As we can see in Figure 1.2, sharing an application implies to upload a taylorized set of information about it into the repository, but it does not imply to send any kind of awareness information as in the case of publishing. In the same way, "retrieving"[1] an application from the repository implies to get it and taylorized it, but not to subscribe to it. On the other hand, once an user has appropriated and adapted the application, he will be able to publish it as one made from the scratch.

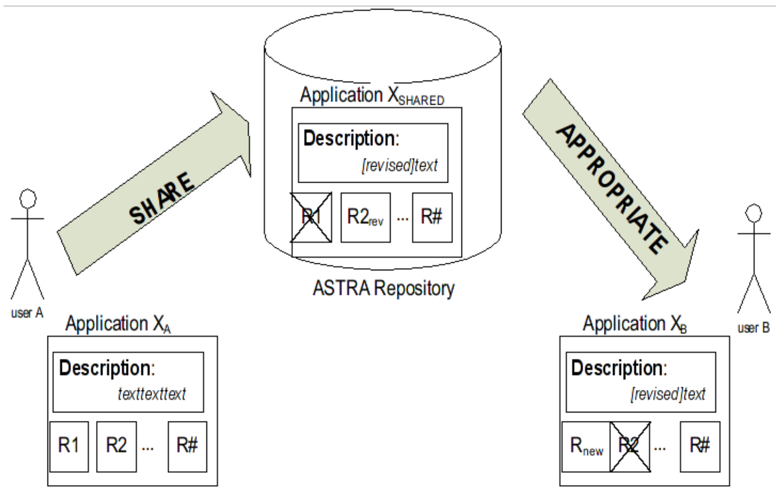


Figure 1.3: Sharing and retrieving applications through the repository

The rest of this document describes the process carried on to develop the previously mentioned system, including a goals statement in Chapter 2, a explanation of the employed methodology and the involved technologies in Chapter 3, a detailed description of the project in Chapter 4, and a discussion of the fulfillment of the goals and the personal contribution in Chapter 5.

[1]We will use the verbs "get", "appropriate" or "retrieve" as synonyms when referring to an application in this document.

# Chapter 2

# Objectives

In this section we will state briefly the main goals of the project. The way in which these objectives were achieved will be discussed in Chapter 4.

The main objectives of this project are:

- Create a main repository for applications, where the users can browse, share and retrieve them. This should be accomplished taking into account:
  - The sharing process has to be flexible enough to allow the user choosing in which communities the application is going to be shared, and which rules are going to be shared.
  - The retrieving process has to be flexible enough to allow the user a customization of the retrieved application.
  - The changes needed to adapt the application have to be as transparent to the user as possible.
  - It is necessary to implement a mechanism which allows the user to search for applications:
    - By different criteria: tags, description, type, etc.
    - Using a system recommendation process, where the user just has to select an application and some similar applications will be suggested.
- Create a system which allows the users to tag the applications. This should be accomplished taking into account the need of different scopes for the tags:
  - Private tags, which are only for personal purposes and should not be stored in the Backend.

  - Community tags, which are only visible for members of that community.
  - Public tags, which are visible for all the members.

- Create a GUI which allows the user to carry out the operations defined previously. This should be performed taking into account:
  - The GUI has to be connected with the rest of systems in a loose coupling way.
  - It has to be intuitive.
  - It has to be extensible, so other systems can be connected to it in the future.

# Chapter 3

# Methodology and involved technologies

## 3.1 Methodology

Due to the researching nature of the main project, it was quite difficult to establish a fixed set of requirements in the beginning. Therefore, we used the following system to carry on the process:

- Establish a set of main objectives which can be expanded afterwards. The final result is the set of objectives which was discussed in Chapter 2.
- Create a set of use cases based on these discussions.
- Create a design which has to be flexible enough to allow the introduction of new changes in the future.
- Implement a prototype and perform tests.
- Do a demonstration.
- New elicitation requirements process if necessary.

This model follows a spiral model (see Figure 3.1), a software development process combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts.

Due to the limited extension of this document, it is not possible to explain in the deserved detail the dynamic followed during the process. But it is important to stand

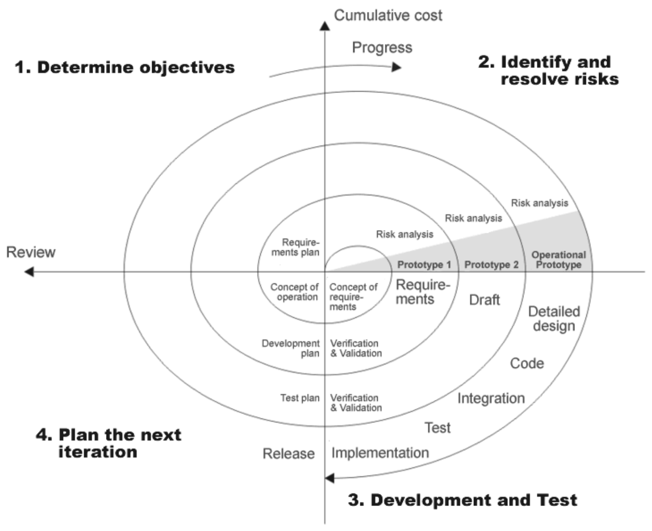


Figure 3.1: Spiral model (Boehm, 1988)

out at least that the whole process took four iterations which are briefly explain in the Table 3.1.

| Iteration | Requirements | Affected components |
|---|---|---|
| 1 | Share and retrieve applications | `RepositoryManager` & `PHP EUT tools` |
| 2 | Tagging | `TagManagerNode` & `TagManagerBackEnd` |
| 3 | Searching capabilities & tagging extensions | `RepositoryManager`, `TagManagerNode` & `TagManagerBackEnd` |
| 4 | GUI | `ApplicationManager` |

Table 3.1: Iterations during the project

We have followed a methodology which can be seen as an Agile software development methodology [San05]. The concept refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. The term was coined in the year 2001 when the Agile Manifesto [Bo01] was formulated.

Agile methods generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices that allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals. Conceptual foundations of this framework are found in modern approaches to operations management and analysis, such as lean manufacturing, soft systems methodology, speech act theory (network of conversations approach), and Six Sigma.

This methodology fits properly with the researching nature of the project and the fact that new requirements arise continuously.

## 3.2 Involved paradigms and technologies

In this section we will explain briefly the main paradigms and technologies involved during the achievement of this project.

### 3.2.1 SOA

Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains [McC08]. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

In computing, the term Service-Oriented Architecture expresses a perspective of software architecture that defines the use of services to support the requirements of software users. In an SOA environment, resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation. SOA can also be regarded as a style of Information Systems architecture that enables the creation of applications that are built by combining loosely coupled and inter-operable services.

The main principles behind the SOA paradigm can be summarized as follows:

- Reuse
- Granularity
- Modularity
- Composability
- Componentization
- Interoperability

- Compliance to standards (both common and industry-specific) (e.g. Web Services)
- Services identification and categorization, provisioning and delivery, and monitoring and tracking

#### 3.2.1.1 SOAP and Apache Axis

SOAP is a lightweight protocol for exchanging structured information in a decentralized, distributed environment [Le 02]. It is an XML (see Section 3.2.5) based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

The implementation we have used is Apache Axis, it consists of an open source Java and C++ implementation of the SOAP server, and various utilities and APIs for generating and deploying Web service applications. Using Apache Axis, it is possible to create interoperable and distributed computing applications. Axis is developed under the auspices of the Apache Software Foundation.

#### 3.2.1.2 ASTRA SOA

The ASTRA Service-Oriented Architecture is the backbone of ASTRA [BP09]. It includes the platform for awareness services, the ontology, ontology management, context management, service discovery, and other necessary modules like the ones developed for this project.

These services are offered by a set of bundles (see Section 3.2.3) that can be grouped into two subsystems from a high level point of view: ASTRA Node and ASTRA Backend, following a Client-Server model[1]. Therefore it is important to distinguish between the local and remote nature connection when consuming other bundles services, to take into account the limitations in the type of objects we can use for our web services interfaces in the case of remote connections, and to threat properly possible problems in the network.

[1]There has been some discussion about the possibility of following a P2P model in ASTRA, but this is out of the scope of the current implementation.

Below are listed the bundles whose services have been used by the bundles developed for this project. The dependencies with them are explained in Sections 4.3.2.1, 4.3.3.1, 4.3.4.1 and 4.3.5.1.

- `UserManager`: It is the responsible for managing users and their profiles, as well as the management of the user identities. It is executed in the Backend.
- `CommunityManager`: It is the responsible for providing the possibility to define, share and connect virtual community representations in within which users can share awareness information. It is executed in the Backend.
- `AwarenessManager`: It is the responsible for the connection between low level user-system interaction and the high level concepts related to them. It is connected with the Rules engine kernel. It is executed in the Nodes.
- `AwarenessApplicationManager`: It is the responsible for storing and managing the local awareness applications. It is executed in the Nodes.
- `OntologyManager`: It is the responsible for managing, looking-up and extending the ontologies. It is executed in the Nodes.
- `PersistencyManager`: It is the responsible for providing storage functionalities. It is executed in the Backend and the Nodes.
- `RemoteFrameworkManager`: It is the responsible for providing facilities to consume remote bundles services. It is executed in both subsystems.
- `EventsManager`: It is the responsible for providing facilities to communicate events between the bundles. It is executed in both subsystems.

### 3.2.2 Java

Most of the development tasks during the project were coded using Java. Java is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture.

The original and reference implementation Java compilers, virtual machines, and class libraries were developed by Sun from 1995. As of May 2007, in compliance with the specifications of the Java Community Process, Sun made available most of their Java technologies as free software under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java and GNU Classpath.

Java has significant advantages over other languages and environments that make it suitable for just about any programming task.

The advantages of Java are as follows:

- Java is easy to learn: Java was designed to be easy to use and is therefore easier to write, compile, debug, and learn than other programming languages.
- Java is object-oriented: This allows us to create modular programs and reusable code.
- Java is platform-independent: One of the most significant advantages of Java is its ability to move easily from one computer system to another. The ability to run the same program on many different systems is crucial to World Wide Web software, and Java succeeds at this by being platform-independent at both the source and binary levels.

Because of Java's robustness, ease of use, cross-platform capabilities and security features, it has become a language of choice for providing worldwide Internet solutions.

### 3.2.3 OSGi

OSGi (Open Services Gateway initiative) is a flexible framework, which provides a standardized environment for service deployment and operation. The Framework implements an elegant, complete, and dynamic component model; something that is missing in standalone Java/VM environments. The platform is java-based and can be remotely managed. In Figure 3.2 we can see the OSGi layered model.

Applications or components (coming in the form of bundles for deployment) can be remotely installed, started, stopped, updated and uninstalled without requiring a reboot. Bundles are deployed on an OSGi framework, the bundle runtime environment. This is not a container like Java Application Servers. It is a collaborative

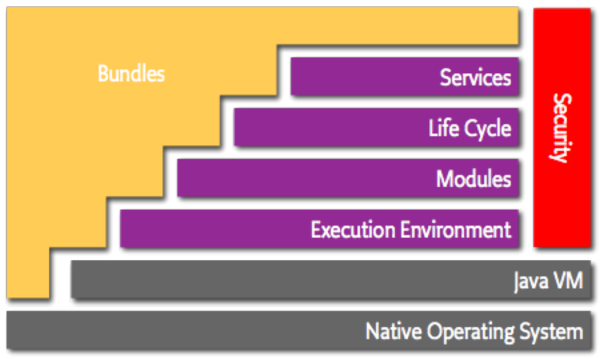


Figure 3.2: OSGi layered model

environment. Bundles run in the same VM and can actually share code. The framework uses the explicit imports and exports to wire up the bundles so they do not have to concern themselves with class loading.

The original focus was on service gateways but the applicability turned out to be much wider. Implementations of the OSGi framework specification are available for a range of different environments and device classes, from backend systems via desktops to mobile devices. A well-known and popular usage for its flexibility is the Eclipse development framework, which is completely based on OSGi. The longest history can OSGi claim on embedded systems, where it originally was designed for. Currently Nokia is integrating OSGi technologies into their latest phones. But also big application servers like IBM WebSphere 6.1 start adapting this technology.

#### 3.2.3.1 OSGi-Knopflerfish

Knopflerfish is a non-profit organization, developing OSGi related material. The project provides easy to use open source certified implementation of the OSGi R4 core framework specification, as well as related build tools and applications. Knopflerfish is available under a BSD style license.

#### 3.2.3.2 Why using OSGi in ASTRA?

OSGi enforces a clean service oriented design approach, with a clear distinction between interfaces and implementation. Services are deployed in bundles, which can be installed, updated and removed during the runtime of the framework. OSGi is thus ideally suited to realizing the principles of the SOA paradigm. OSGi also provides an Apache Axis (see Section 3.2.1.1) bundle that allows for automatic generation of web service interfaces.

The decision of using the OSGi framework has several severe impacts on the design of the functional components. The most important implications for the component design are summarized here:

- Services are de-coupled. A service must not take any assumptions about the existence and life-cycle of any other service it might want to use (concept of "minimal assumptions").
- Services communicate only through their exposed interfaces.
- A service must not take any assumptions about the implementation details of any other service. The smallest unit for code deployment is a bundle, which might simply contain API's or one or more service implementations.

It is worth stressing that OSGi is chosen as a platform, but it is in no way a requirement by ASTRA to use OSGi. It is used because it offers a flexible and convenient way of deploying the ASTRA SOA functional components. Any ASTRA component can be developed outside OSGi, and made available through a Web Service interface.

### 3.2.4 Swing

Swing is a widget toolkit for Java. It is part of Sun Microsystems Java Foundation Classes (JFC), an API for providing a graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit. It provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform.

Swing has been used to develop a bundle which implements a GUI to allow the user to perform all the operations related to the applications and tags management process. The reason why Swing was chosen is the set of features that its architecture provides:

- Platform independence: Swing is platform independent both in terms of its expression (Java) and its implementation (non-native universal rendering of widgets).

- Extensibility: Swing is a highly partitioned architecture, which allows for the "plugging" of various custom implementations of specified framework interfaces: Users can provide their own custom implementation(s) of these components to override the default implementations. In general, Swing users can extend the framework by extending existing (framework) classes and/or providing alternative implementations of core components.

- Component-oriented: Swing is a component-based framework. The distinction between objects and components is a fairly subtle point: concisely, a component is a well-behaved object with a known/specified characteristic pattern of behaviour. Swing objects asynchronously fire events, have "bound" properties, and respond to a well-known set of commands (specific to the component). Specifically, Swing components are Java Beans components, compliant with the Java Beans Component Architecture specifications.

- Customizable: Given the programmatic rendering model of the Swing framework, fine control over the details of rendering of a component is possible in Swing. As a general pattern, the visual representation of a Swing component is a composition of a standard set of elements, such as a "border", "inset", decorations, etc. Typically, users will programmatically customize a standard Swing component (such as a `JTable`) by assigning specific Borders, Colors, Backgrounds, opacities, etc., as the properties of that component. The core component will then use these property (settings) to determine the appropriate renderers to use in painting its various aspects. However, it is also completely possible to create unique GUI controls with highly customized visual representation.

- Configurable: Swing's heavy reliance on runtime mechanisms and indirect composition patterns allows it to respond at runtime to fundamental changes in its settings. For example, a Swing-based application can change its look and feel at runtime. Further, users can provide their own look and feel implementation, which allows for uniform changes in the look and feel of existing Swing applications without any programmatic change to the application code.

These traits allowed us to fulfill successfully the requirements for the above-mentioned bundle, which will be explained deeply in Sections 4.3.5 and 4.4.2.

### 3.2.5 XML

XML (eXtensible Markup Language) is a general-purpose specification for creating custom markup languages. It is classified as an extensible language, because it allows the user to define the mark-up elements.

XML's purpose is to aid information systems in sharing structured data, especially via the Internet, to encode documents, and to serialize data; in the last context, it compares with text-based serialization languages such as JSON, YAML, and S-Expressions.

XML's set of tools helps developers in creating web pages but its usefulness goes well beyond that. XML, in combination with other standards, makes it possible to define the content of a document separately from its formatting, making it easy to reuse that content in other applications or for other presentation environments. Most importantly, XML provides a basic syntax that can be used to share information between different kinds of computers, applications and organizations without needing to pass through many layers of conversion.

XML began as a simplified subset of the Standard Generalized Markup Language (SGML), meant to be readable by people via semantic constraints. Application languages can be implemented in XML. These include XHTML, RSS, MathML, GraphML, Scalable Vector Graphics, MusicXML, and others. Moreover, XML is sometimes used as the specification language for such application languages.

XML is recommended by the World Wide Web Consortium (W3C). It is a fee-free open standard. The recommendation specifies lexical grammar and parsing requirements.

As we will explain in detail in Section 4.4.4, XML is the language used to represent the rules in ASTRA.

### 3.2.6 DOM

The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and

XML documents. Objects under the DOM (also sometimes called "Elements") may be specified and addressed according to the syntax and rules of the programming language used to manipulate them. The rules for programming and interacting with the DOM are specified in the DOM Application Programming Interface (API).

As we will explain in detail in Section 4.4.4, DOM was used to access and modify the rules in ASTRA.

### 3.2.7 Lucene

Apache Lucene is an open source information retrieval library, originally created in Java by Doug Cutting. It is supported by the Apache Software Foundation and is released under the Apache Software License. Lucene itself is just an indexing and search library and does not contain crawling and HTML parsing functionality: it is not an application. It allowed us to create a search engine integrated in one of the bundles (see Sections 4.3.2 and 4.4.3) to offer searching capabilities in it. The main reasons why Lucene was proposed (and accepted) were:

- It is open-source.
- It has a great performance.
- It is quite flexible and easy to extend if more functionalities are needed in the future.
- It is cross-platform.

### 3.2.8 MySQL

MySQL is a relational database management system (RDBMS) very popular in the free and open source software communities. The program runs as a server providing multi-user access to a number of databases.

The project's source code is available under terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL is owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now a subsidiary of Sun Microsystems, which holds the copyright of most of the code.

MySQL is commonly used by free software projects which require a full-featured database management system, such as WordPress, phpBB and other software built on the LAMP software stack. It is also used in very high-scale World Wide Web products including Google and Facebook.

MySQL is the RDBMS system which is used for taking care of the persistence in ASTRA.

# Chapter 4

# Description

In this chapter we will detail the process carried on to transform the initial objectives into requirements, and the design and development of the application based on them.

## 4.1 Requirements

In this section we will describe shortly the requirements that we gathered during the several requirements elicitation processes performed during all the iterations (see Table 3.1). It is interesting to remark that this process becomes even more important (and challenging) in projects of a researching nature like ASTRA, due to the continuous rise of requirements implicit in its kind.

Tables 4.1 and 4.2 summarize the most important functional and non functional requirements that we gathered respectively.

| **Functional requirements** | |
|---|---|
| 1. | Create a repository to store awareness applications. |
| 2. | The information to share can be customized by the user before being stored. |
| 3. | The repository has to take into account the visibility of the applications in terms of communities. |
| 4. | It is necessary to offer functionalities to browse the repository by communities. |
| 5. | It is necessary to create a mechanism to search applications by different criteria (tags, description, type or any). |
| 6. | It is necessary to create a mechanism to recommend applications based on the similarity with respect to another application. |
| 7. | During the appropriation of an application, it is necessary to offer the user the possibility of customizing the application (i.e.: choosing the rules). |
| 8. | Create a mechanism to obtain a “human readable” description of a rule. |
| 9. | Create a system which allows us to tag applications choosing the visibility: public, communities or private. |
| 10. | Create a GUI which allows the interaction with the repository and perform common operations (logging in, showing help, etc.). |

Table 4.1: Functional requirements

| **Non functional requirements** | |
|---|---|
| 1. | The functionalities have to be offered by OSGi bundles through their web services interfaces. |
| 2. | The system has to perform the retrieving process as transparent as possible for the user. |
| 3. | All the components have to be multiplatform. |
| 4. | The GUI has to be intuitive. |
| 5. | The GUI has to be easy to connect to other bundles in the future. |
| 6. | It is necessary to study the possibility of using ontology services provided by `OntologyManager` in order to improve the performance of some of the functionalities (i.e.: application adaptation). |

Table 4.2: Non functional requirements

## 4.2 Use cases

In this section we will state the main use cases identified and we detail the interactions of the main scenarios for each of them.

### 4.2.1 Repository management

The Figure 4.1 shows the main use cases related with the repository management process, which are detailed in Tables 4.3, 4.4 and 4.5.

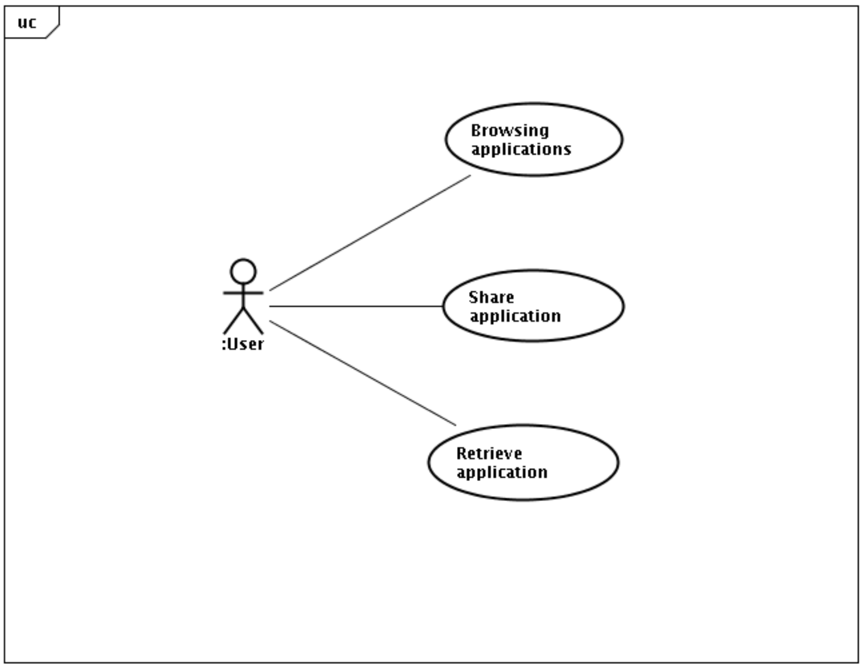


Figure 4.1: Repository management use cases

| **Browse applications** | |
|---|---|
| Brief description: | Actor browses repository applications. |
| Actors: | Authenticated ASTRA user. |
| Preconditions: | There are applications visible for that user. |
| Basic flow of events: | |
| 1. | User expands the applications tree. |
| 2. | User selects an application. |
| 3. | System retrieves all the information related to it. |
| 4. | System displays the information. |
| Postconditions: | - |

Table 4.3: Browse applications - general scenario description

| **Share application** | |
|---|---|
| Brief description: | Actor shares an application in the repository. |
| Actors: | Authenticated ASTRA user. |
| Preconditions: | A local application has been selected. |
| Basic flow of events: | |
| 1. | User selects share application. |
| 2. | System displays all the information related to it. |
| 3. | User customizes the parameters to be shared: |
| | visibility in terms of communities, rules, description, etc. |
| 4. | User confirms the sharing process. |
| 5. | System validates the parameters. |
| 6. | System confirms the application was shared properly. |
| Postconditions: | A new application is available in the repository. |

Table 4.4: Share application - general scenario description

| **Retrieve application** | |
|---|---|
| Brief description: | Actor retrieves an application from the repository. |
| Actors: | Authenticated ASTRA user. |
| Preconditions: | An application from the repository has been selected. |
| Basic flow of events: | |
| 1. | User selects retrieve application. |
| 2. | System displays all the information related to it. |
| 3. | User customizes the parameters to be retrieved: rules and description. |
| 4. | User confirms the retrieving process. |
| 5. | System validates the parameters. |
| 6. | System confirms the application was retrieved properly. |
| Postconditions: | A new application is available in the user local space. |

Table 4.5: Retrieve application - general scenario description

### 4.2.2 Searching

The Figure 4.2 shows the main use cases related with the searching requirements, which are detailed in Tables 4.6 and 4.7.

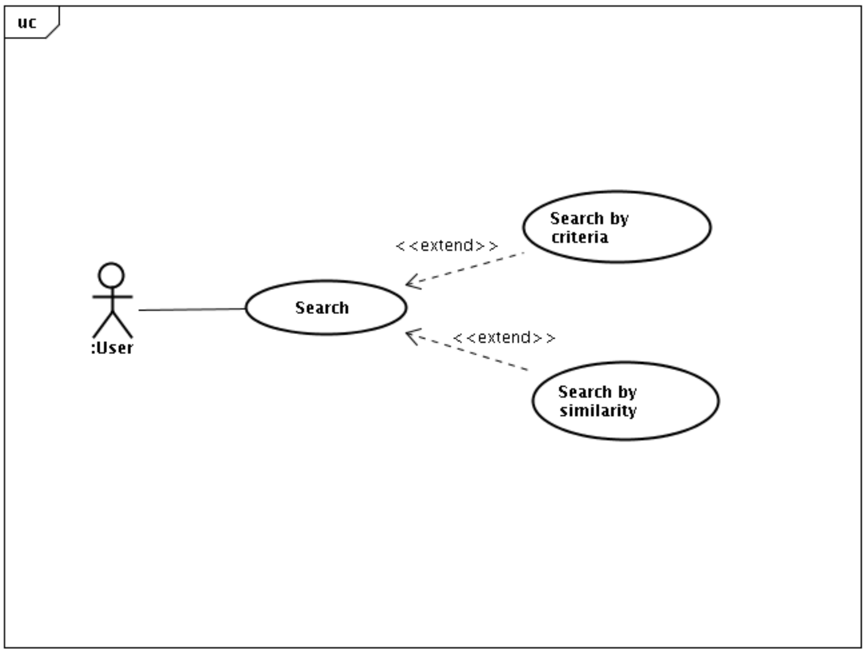


Figure 4.2: Searching use cases

| Searching by criteria | |
|---|---|
| Brief description: | Actor searches for an application typing some keywords and a criterion. |
| Actors: | Authenticated ASTRA user. |
| Preconditions: | - |
| Basic flow of events: | |
| 1. | User enters one or more keywords. |
| 2. | User selects a criterion (by tags, by description, by type or any). |
| 3. | System performs a search based on the given parameters. |
| 4. | System displays matching applications. |
| Postconditions: | - |

Table 4.6: Search by criteria - general scenario description

| **Searching by similarity** | |
|---|---|
| Brief description: | Actor searches for applications that are similar to a given application. |
| Actors: | Authenticated ASTRA user. |
| Preconditions: | - |
| Basic flow of events: | |
| 1. | User selects an application. |
| 2. | System compares the selected application with applications in the repository using different measures. |
| 3. | System displays matching applications. |
| Postconditions: | - |

Table 4.7: Search by similarity - general scenario description

### 4.2.3 Tags management

The Figure 4.3 shows the main use cases related with the tags management process, which are detailed in Tables 4.8 and 4.9.

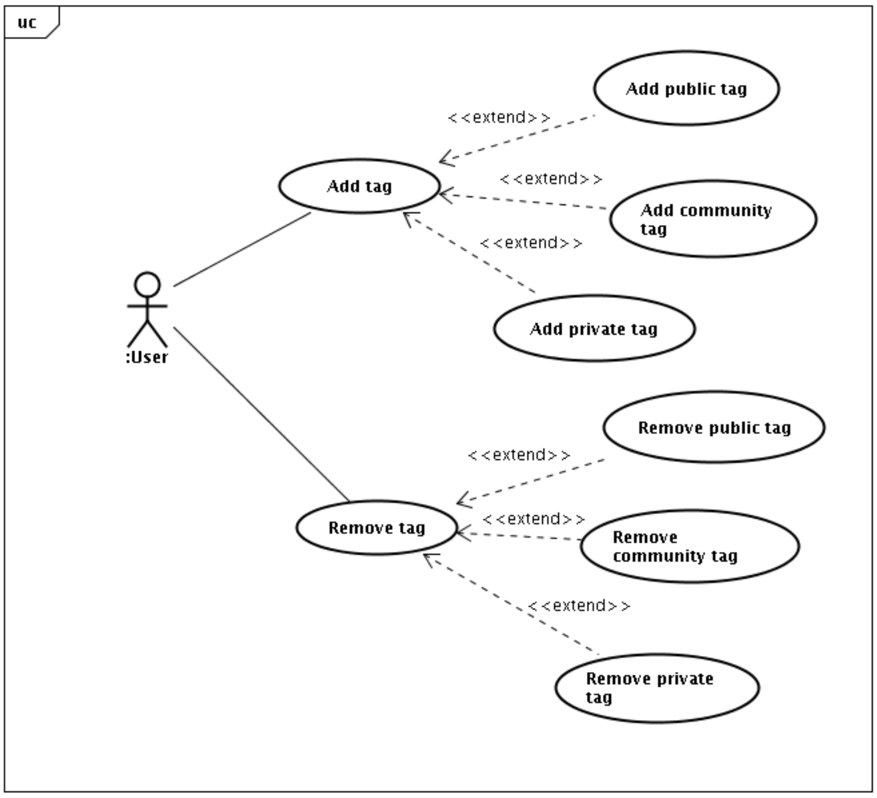


Figure 4.3: Tagging use cases

| **Add tag** | |
|---|---|
| Brief description: | Actor adds a tag to an application specifying the level of visibility of the tag. |
| Actors: | Authenticated ASTRA user. |
| Preconditions: | An application has been selected. |
| Basic flow of events: | |
| 1. | User enters a tag name. |
| 2. | User specifies tag visibility. |
| 3. | System validates tag. |
| 4. | System stores the tag. |
| 5. | System confirms that tag has been added. |
| Postconditions: | A new tag is associated to the application and is made available within the proper scope. |

Table 4.8: Add tag - general scenario description

| **Remove tag** | |
|---|---|
| Brief description: | Actor removes a tag from an application. |
| Actors: | Authenticated ASTRA user. |
| Preconditions: | User has previously tagged the selected application. |
| Basic flow of events: | |
| 1. | User selects a tag. |
| 2. | System displays option to remove the tag. |
| 3. | User chooses to remove the tag. |
| 4. | System deletes the tag. |
| 5. | System confirms that tag has been deleted. |
| Postconditions: | The specified tag is no longer associated with the specified application. |

Table 4.9: Remove tag - general scenario description

### 4.2.4 Other general functionalities

The Figure 4.4 shows use cases that capture other general functionalities, which are detailed in Tables 4.10, 4.11 and 4.12.

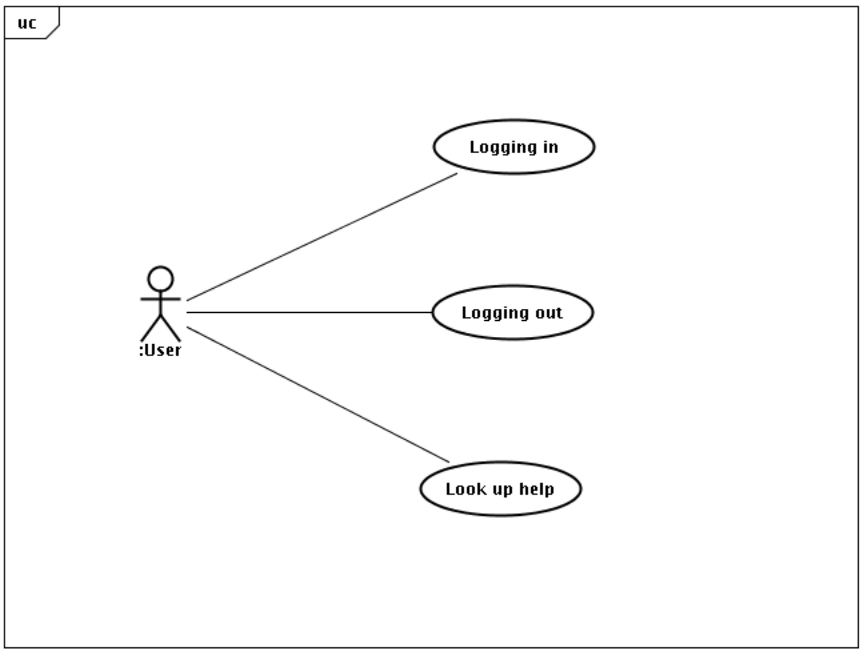


Figure 4.4: Other general functionalities

| Logging in | |
|---|---|
| Brief description: | Actor logs in the system. |
| Actors: | Non-authenticated ASTRA user. |
| Preconditions: | Login window is displayed. |
| Basic flow of events: | |
| 1. | User types username and password. |
| 2. | System validates them. |
| 3. | System loads user's profile. |
| 4. | System displays the main menu and disposes the login window. |
| Postconditions: | - |

Table 4.10: Logging in - general scenario description

| **Logging out** | |
|---|---|
| Brief description: | Actor logs out of the system. |
| Actors: | Authenticated ASTRA user. |
| Preconditions: | Main window is displayed. |
| Basic flow of events: | |
| 1. | User selects logout option. |
| 2. | System asks for confirmation. |
| 3. | User confirms. |
| 4. | System closes user's session, disposes the main window and displays the login window |
| Postconditions: | - |

Table 4.11: Logging out - general scenario description

| **Look up help** | |
|---|---|
| Brief description: | Actor looks up the online help. |
| Actors: | Authenticated ASTRA user. |
| Preconditions: | - |
| Basic flow of events: | |
| 1. | User selects online help option. |
| 2. | System retrieves the remote help. |
| 3. | System displays the help contents in an intuitive way. |
| Postconditions: | - |

Table 4.12: Look up help - general scenario description

## 4.3 Design

In this section we will explain the most important design decisions and the reasons why they were taken. We will also go into detail in the most remarkable characteristics.

### 4.3.1 Bundles design

One of the first and most important decisions is how to divide the functionality into components. Having the requirement of using OSGi, it seems natural to use a bundle as the notion of component. The final design consists of four bundles:

- `RepositoryManager`, which has the following responsibilities:
  - Offer services to share applications.
  - Offer services to retrieve applications.
  - Storage of shared applications.
  - Offer services to search applications by criteria.
  - Offer services to search applications by similarity.
- `TagManagerBackEnd`, which has the following responsibilities:
  - Offer services to add and delete public and community tags.
  - Offer services to retrieve those tags in different and flexible ways.
- `TagManagerNode`, which has the following responsibilities:
  - Offer services to add and delete private tags.
  - Offer services to retrieve those tags in different and flexible ways.
- `ApplicationManager`, which is responsible of the interaction with the user, and connects with the proper bundles to satisfy his requests.

The components diagram in Figure 4.5 shows the connection between all of them through its interfaces[1]:

Sections 4.3.2, 4.3.3, 4.3.4 and 4.3.5 explain the most important features of each of them.

[1]The connection with the rest of ASTRA bundles is omitted for simplicity reasons, but it will be discussed in the section where every bundle is explained.

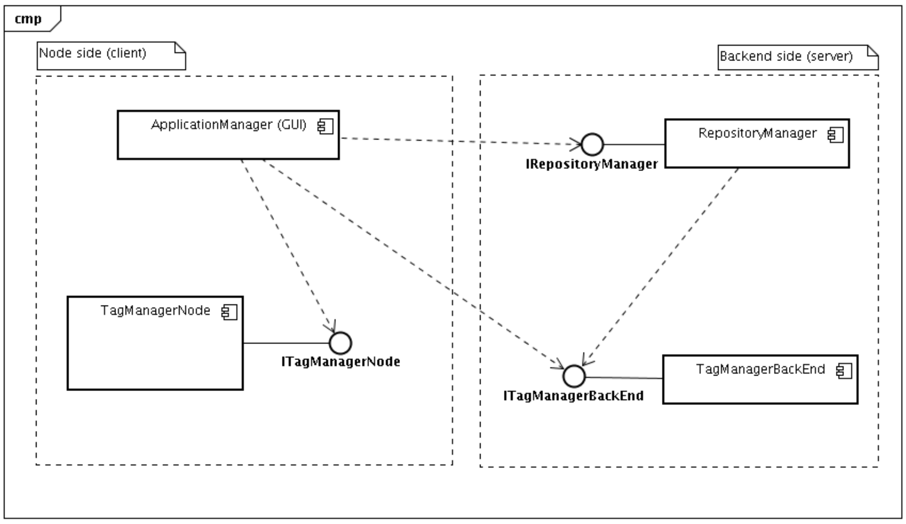


Figure 4.5: Relationships between the bundles developed for this project (components diagram)

### 4.3.2 RepositoryManager

Since one of the most important goals of the project is the need of allowing the users to share and retrieve applications, `RepositoryManager` is one of the key pieces of the project. This bundle is executed in the Backend (server-side), and it offers a clear interface to make its services available for the rest of the bundles in ASTRA.

In this section we will explain the process we followed to perform its design. Some of the trickiest implementation details about this bundle are explained in Section 4.4.

#### 4.3.2.1 Connection with other bundles

The first step consists of analyzing the relationship between this bundle and the rest of bundles in ASTRA. Taking into account the requirements stated in Section 4.1 and the analysis performed in Sections 4.2.1 and 4.2.2, we needed to make use of the services of the following bundles:

- `CommunityManager`: Necessary to retrieve information about the relationship between users, their communities and the applications. I.e.: to assure the visibility of certain application taking into account the communities joined for an user who is going to retrieve applications.

- `TagManagerBackEnd`[2]: Necessary to analyze the tags in order to construct the index of the search engine.
- `EventsManager`: Necessary to keep track of the events produced in `TagManagerBackEnd`, in order to keep the index of the search engine updated. I.e: new tags or tags that have been deleted.
- `PersistencyManager`: Necessary to store permanently all the data into the database. Using its services, we assure the robustness of the system.

The Figure 4.6 shows graphically the relationship between `RepositoryManager` and the rest of components through its interfaces.

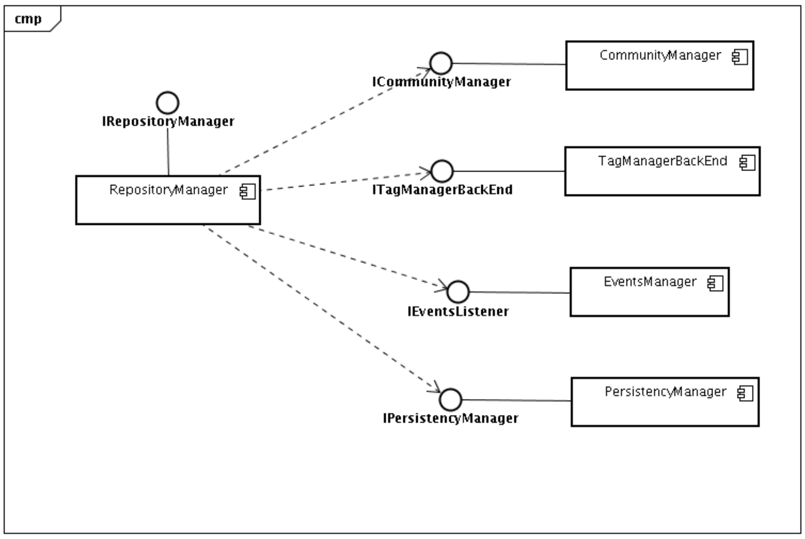


Figure 4.6: RepositoryManager (components diagram)

#### 4.3.2.2 Interface definition

The next step carried on was defining an interface. The Tables 4.13, 4.14, 4.15 and 4.16 summarize the methods offered by the final version of `RepositoryManager`.

[2]The design of this bundle is explained in Section 4.3.3

| **createSharedApplication** | | |
|---|---|---|
| Description: Create a new shared application in the repository. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| userId | String | Identifier of the user who uploads the application. |
| appType | String | Type of application. |
| appDescription | String | Application description. |
| Returns: Boolean, true if everything is ok, false if there was a failure. | | |
| **deleteSharedApplication** | | |
| Description: Delete a SharedApplication in the repository. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format |
| Returns: Boolean, true if the application was deleted properly, false otherwise. | | |
| **shareInCommunity** | | |
| Description: Store relationship between a shared application and a community. | | |
| Parameter | Type | Description |
| userId | String | User identifier in the standard format. |
| appId | String | Application identifier in the standard ASTRA format. |
| communityId | String | Community identifier in the standard format. |
| Returns: Boolean, true if the operation was successful, false otherwise. | | |
| **createSharedRule** | | |
| Description: Create a new shared rule associated to the application. | | |
| Parameter | Type | Description |
| userId | String | User identifier in the standard format. |
| appId | String | Application identifier in the standard ASTRA format. |
| ruleId | String | Rule identifier in the standard format. |
| xmlData | String | XML file which contains the rule. |
| Returns: Boolean, true if the operation was successful, false otherwise. | | |

Table 4.13: RepositoryManager interface

| **listSharedApplications** | | |
|---|---|---|
| Description: Returns the list of applications in the repository visible for that user. | | |
| Parameter | Type | Description |
| userId | String | User identifier in the standard format. |
| Returns: String array, list of applications. | | |

| **listSharedRules** | | |
|---|---|---|
| Description: Returns the list of rules in the repository associated to appID | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| Returns: String array, list of rule identifiers. | | |

| **getXmlData** | | |
|---|---|---|
| Description: Returns the XML file which describes the rule. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| ruleId | String | Rule identifier in the standard format. |
| Returns: String, XML file describing the rule. | | |

| **getSharedApplicationName** | | |
|---|---|---|
| Description: Returns the name of the application. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| Returns: String, application name. | | |

| **getSharedApplicationOwner** | | |
|---|---|---|
| Description: Returns the owner of the application. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| Returns: String, application owner. | | |

| **getSharedApplicationDescription** | | |
|---|---|---|
| Description: Returns the description of the application. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| Returns: String, application description. | | |

Table 4.14: RepositoryManager interface (II)

| **getSharedApplicationDate** | | |
|---|---|---|
| Description: Returns the date of the application. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| Returns: String, application date. | | |

| **getSharedApplicationType** | | |
|---|---|---|
| Description: Returns the type of the application. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| Returns: String, application type. | | |

| **isAlreadyShared** | | |
|---|---|---|
| Description: Checks if an application has already been shared. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| Returns: Boolean, true if already exists, false otherwise. | | |

| **getXmlRuleDescription** | | |
|---|---|---|
| Description: Returns a description of the rule in a readable way using the XML file information. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| ruleId | String | Rule identifier in the standard format. |
| Returns: String, description of the rule. | | |

| **getXmlRule** | | |
|---|---|---|
| Description: Returns the XML file which describes the rule with the ownership already modified. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| ruleId | String | Rule identifier in the standard format. |
| newUserId | String | New user identifier in the standard format. |
| Returns: String, XML file associated to the rule. | | |

Table 4.15: RepositoryManager interface (III)

| **search (overloaded)** | | |
|---|---|---|
| Description: It performs a search according to one criterion. | | |
| Parameter | Type | Description |
| userId | String | Identifier of the user who is performing the search. |
| q | String | Query. |
| criterion | String | Criterion to use. |
| Returns: String array, with the identifiers of the matching applications. | | |
| **search (overloaded)** | | |
| Description: It performs a search according to a set of criteria. | | |
| Parameter | Type | Description |
| userId | String | Identifier of the user who is performing the search. |
| q | String | Query. |
| criteria | String array | Criteria to use. |
| Returns: String array, with the identifiers of the matching applications. | | |
| **searchBySimilarity** | | |
| Description: It performs a search of similar applications. | | |
| Parameter | Type | Description |
| userId | String | Identifier of the user who is performing the search. |
| appId | String | Application identifier in the standard ASTRA format. |
| appDescription | String | Application description. |
| Returns: String array, with the identifiers of the matching applications. | | |

Table 4.16: RepositoryManager interface (IV)

#### 4.3.2.3 Classes definition

The functionalities of this bundle are structured in a set of classes from which we will remark:

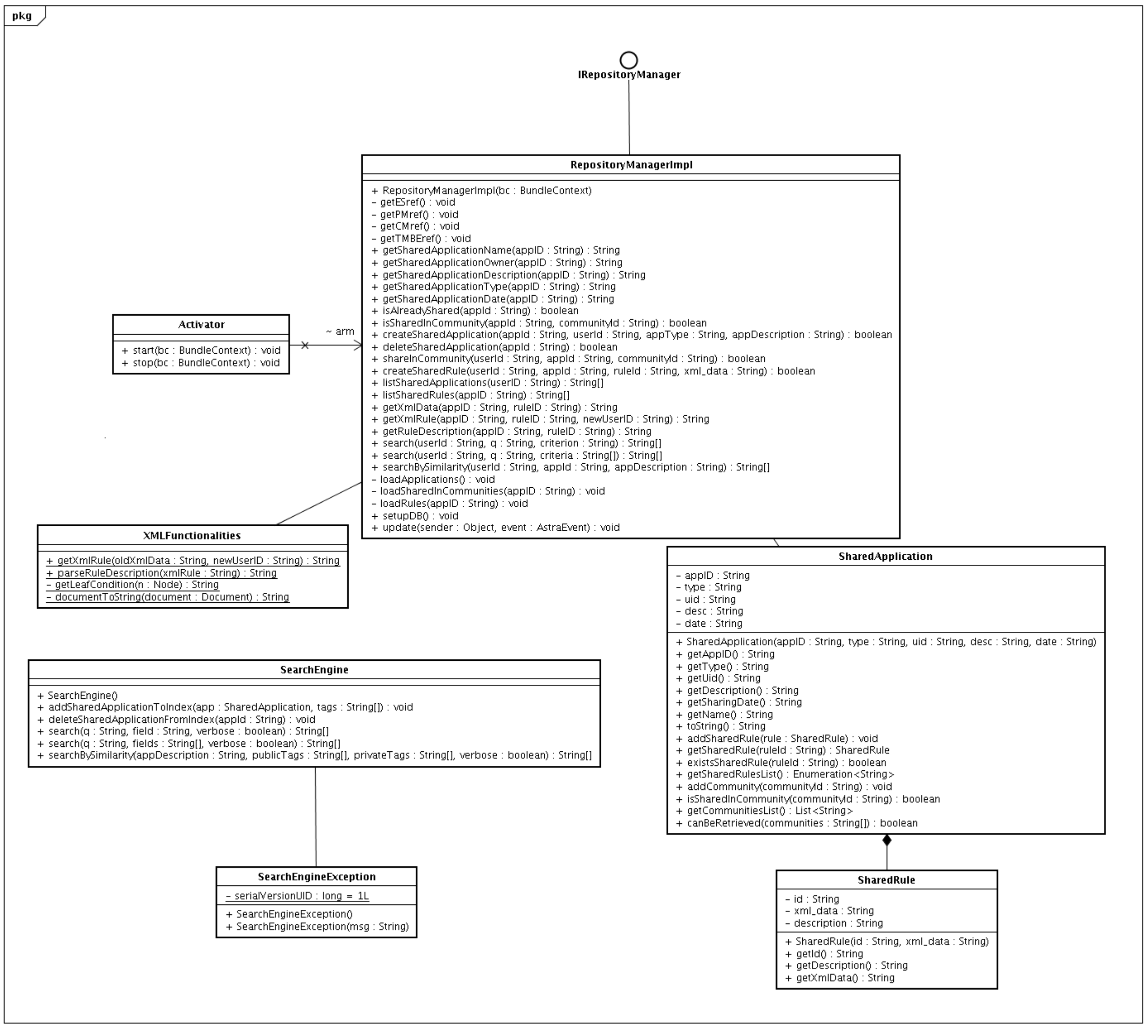
- `RepositoryManagerImpl`:
  - It implements all the methods of `IRepositoryManager`.
  - It manages the connection with the rest of the bundles.
  - It takes care of the integration with the search engine.
- `SharedApplication`: It represents an application in the repository. It is composed by `SharedRules`.
- `SharedRule`: It represents a shared rule in the repository.
- `SearchEngine`: It implements a search engine to perform queries about the applications in the repository.
- `XMLFunctionalities`: Class which provides auxiliar abstract methods related to the XML functionalities (i.e. method to create a rule description receiving the XML file).

The Figure 4.7 shows the most important classes with its main methods, and the relationship between them.

#### 4.3.2.4 Storage system

Finally, we needed to design a database model for `RepositoryManager`. We designed it taking into account the following relationships:

- An user owns (has shared) `0..n` applications in the repository.
- A shared application has `1..n` shared rules.
- A shared application is visible for `1..n` communities.
- A community has `0..n` shared applications.
- A shared application has `0..n` tags.



```
RepositoryManagerImpl
+ RepositoryManagerImpl(bc : BundleContext)
- getESref() : void
- getPMref() : void
- getCMref() : void
- getTMBEref() : void
+ getSharedApplicationName(appID : String) : String
+ getSharedApplicationOwner(appID : String) : String
+ getSharedApplicationDescription(appID : String) : String
+ getSharedApplicationType(appID : String) : String
+ getSharedApplicationDate(appID : String) : String
+ isAlreadyShared(appId : String) : boolean
+ isSharedInCommunity(appId : String, communityId : String) : boolean
+ createSharedApplication(appId : String, userId : String, appType : String, appDescription : String) : boolean
+ deleteSharedApplication(appId : String) : boolean
+ shareInCommunity(userId : String, appId : String, communityId : String) : boolean
+ createSharedRule(userId : String, appId : String, ruleId : String, xml_data : String) : boolean
+ listSharedApplications(userID : String) : String[]
+ listSharedRules(appID : String) : String[]
+ getXmlData(appID : String, ruleID : String) : String
+ getXmlRule(appID : String, ruleID : String, newUserID : String) : String
+ getRuleDescription(appID : String, ruleID : String) : String
+ search(userId : String, q : String, criterion : String) : String[]
+ search(userId : String, q : String, criteria : String[]) : String[]
+ searchBySimilarity(userId : String, appId : String, appDescription : String) : String[]
- loadApplications() : void
- loadSharedInCommunities(appID : String) : void
- loadRules(appID : String) : void
+ setupDB() : void
+ update(sender : Object, event : AstraEvent) : void

Activator
+ start(bc : BundleContext) : void
+ stop(bc : BundleContext) : void

XMLFunctionalities
+ getXmlRule(oldXmlData : String, newUserID : String) : String
+ parseRuleDescription(xmlRule : String) : String
- getLeafCondition(n : Node) : String
- documentToString(document : Document) : String

SearchEngine
+ SearchEngine()
+ addSharedApplicationToIndex(app : SharedApplication, tags : String[]) : void
+ deleteSharedApplicationFromIndex(appId : String) : void
+ search(q : String, field : String, verbose : boolean) : String[]
+ search(q : String, fields : String[], verbose : boolean) : String[]
+ searchBySimilarity(appDescription : String, publicTags : String[], privateTags : String[], verbose : boolean) : String[]

SearchEngineException
- serialVersionUID : long = 1L
+ SearchEngineException()
+ SearchEngineException(msg : String)

SharedApplication
- appID : String
- type : String
- uid : String
- desc : String
- date : String
+ SharedApplication(appID : String, type : String, uid : String, desc : String, date : String)
+ getAppID() : String
+ getType() : String
+ getUid() : String
+ getDescription() : String
+ getSharingDate() : String
+ getName() : String
+ toString() : String
+ addSharedRule(rule : SharedRule) : void
+ getSharedRule(ruleId : String) : SharedRule
+ existsSharedRule(ruleId : String) : boolean
+ getSharedRulesList() : Enumeration<String>
+ addCommunity(communityId : String) : void
+ isSharedInCommunity(communityId : String) : boolean
+ getCommunitiesList() : List<String>
+ canBeRetrieved(communities : String[]) : boolean

SharedRule
- id : String
- xml_data : String
- description : String
+ SharedRule(id : String, xml_data : String)
+ getId() : String
+ getDescription() : String
+ getXmlData() : String
```

Figure 4.7: RepositoryManager (class diagram)

The Figure 4.8 shows a database diagram with all the necessary tables and the relationships between them which were previously enumerated.

It is also important to remark that in order to increase the performance in querying, the repository keeps also a copy in memory of all the information about the applications and the rules stored. This information is stored in hash tables that are synchronized with the data in the database.
This decision introduces as a drawback the need of duplicating the number of create and delete operations (which have to be executed in memory and in the database) and arises the need of establishing synchronization mechanisms to have consistent copies in both sides. But considering that most of the operations are to retrieve

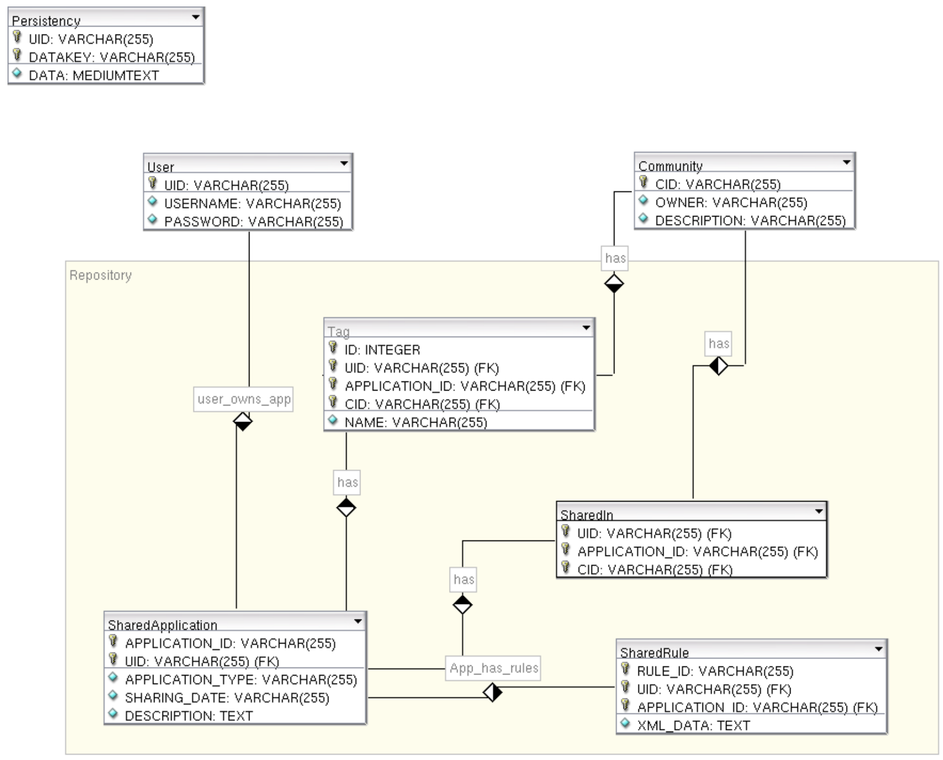


Figure 4.8: RepositoryManager (database model)

information (i.e.: all the needed operations for searching or getting an application) the global performance also increases.

### 4.3.3 TagManagerBackEnd

As it was explained in Section 4.3.1 we decided to divide the tags management into two components: `TagManagerBackEnd` and `TagManagerNode` (which will be explained in Section 4.3.4).

`TagManagerBackEnd` is the bundle that manages the public and community tags. It is executed in the Backend, and it possess an interface to make its services available to the rest of the bundles in ASTRA. The initial version of the code was based on the code for a tagging system under the project Ubicollab[3] by Christian Laverton, which is licensed under an Apache License (version 2.0). It was first adapted for ASTRA,

[3]Ubiquitous Collaboration: is an open source project aiming at implementing a platform for mobile communities developed at NTNU (http://ubicollab.idi.ntnu.no).

and extended afterwards.
In this section we will explain the process we followed to carry on its design, taking a similar approach to the one we took to design `RepositoryManager` (Section 4.3.2).

#### 4.3.3.1 Connection with other bundles

The first step consisted of defining the relationship between `TagManagerBackEnd` and the rest of the bundles in ASTRA.

Taking into account the requirements stated in Section 4.1 and the analysis performed in Section 4.2.3, we needed to make use of the services of the following bundles:

- `CommunityManager`: Necessary to retrieve information about the relationship between users, their communities and the tags. I.e.: to make available certain tag taking into account the scope in terms of visibility of the user.
- `EventsManager`: Necessary to give feedback of the events produced in it. I.e: to inform other bundles that a tag has been added.
- `PersistencyManager`: Necessary to store permanently all the data into the database. Using its services, we assure the robustness of the system.

The Figure 4.9 shows graphically the relationship between `TagManagerBackEnd` and the rest of components through its interfaces.

#### 4.3.3.2 Interface definition

The next step consisted of defining its interface. The Tables 4.17 and 4.18 summarize the methods offered by the final version of `TagManagerBackEnd`.

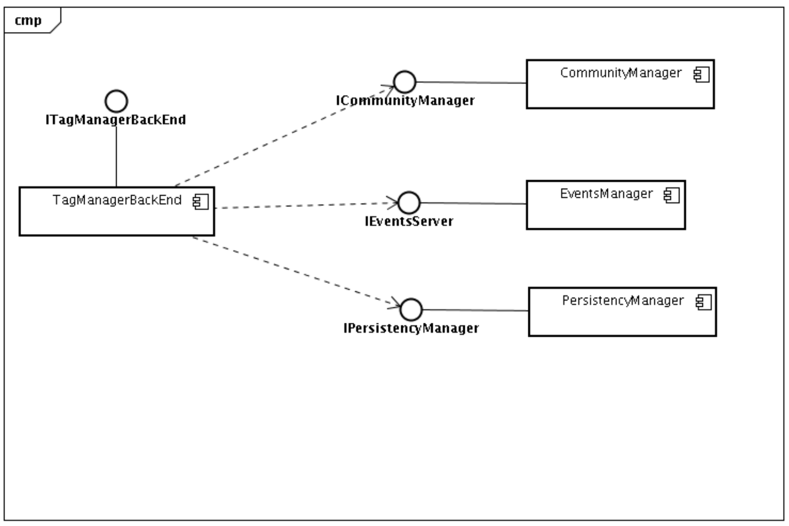


Figure 4.9: TagManagerBackEnd (components diagram)

| | | |
|---|---|---|
| **addTag (overloaded)** | | |
| Description: Creates a public tag. | | |
| Parameter | Type | Description |
| name | String | Tag name. |
| appId | String | Application identifier in the standard ASTRA format. |
| userId | String | Identifier of the user who tags the application. |
| Returns: Boolean, true if the operation was successful, false otherwise. | | |
| **addTag (overloaded)** | | |
| Description: Creates a tag only visible for that community. | | |
| Parameter | Type | Description |
| name | String | Tag name. |
| appId | String | Application identifier in the standard ASTRA format. |
| userId | String | Identifier of the user who tags the application. |
| communityId | String | Community identifier. |
| Returns: Boolean, true if the operation was successful, false otherwise. | | |
| **getTags** | | |
| Description: Returns a list of public tags for the given application. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| limit | Integer | Limit on number of results returned. |
| Returns: String array, list of public tags for the given application. | | |
| **getTagsByCommunity** | | |
| Description: Returns a list of public tags for the given application and community. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| limit | Integer | Limit on number of results returned. |
| communityId | String | Community identifier. |
| Returns: String array, list of public tags for the given application and community. | | |

Table 4.17: TagManagerBackEnd interface

| **getTagsByApplication** | | |
|---|---|---|
| Description: Returns a list of tags associated to the application which are public or visible for that user (because he belongs to that community). | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| userId | String | User identifier. |
| limit | Integer | Limit on number of results returned. |
| Returns: String array, list of visible tags for this user associated to the given application. | | |
| **deleteTag (overloaded)** | | |
| Description: Deletes a public tag. | | |
| Parameter | Type | Description |
| name | String | Tag name. |
| appId | String | Application identifier in the standard ASTRA format. |
| userId | String | Identifier of the user who tagged the application. |
| Returns: Boolean, true if the operation was successful, false otherwise. | | |
| **deleteTag (overloaded)** | | |
| Description: Deletes a tag for a given community. | | |
| Parameter | Type | Description |
| name | String | Tag name. |
| appId | String | Application identifier in the standard ASTRA format. |
| userId | String | Identifier of the user who tagged the application. |
| communityId | String | Community identifier. |
| Returns: Boolean, true if the operation was successful, false otherwise. | | |

Table 4.18: TagManagerBackEnd interface (II)

#### 4.3.3.3 Classes definition

The definition of the classes in this case is simpler than the one explained in Section 4.3.2.3: the main functionalities are provided by the class `TagManagerBackEnd`, which implements the methods of the interface `ITagManagerBackEnd` explained before in Section 4.3.3.2 and is responsible for the communication with the bundles.
The Figure 4.10 shows the class diagram for this bundle.

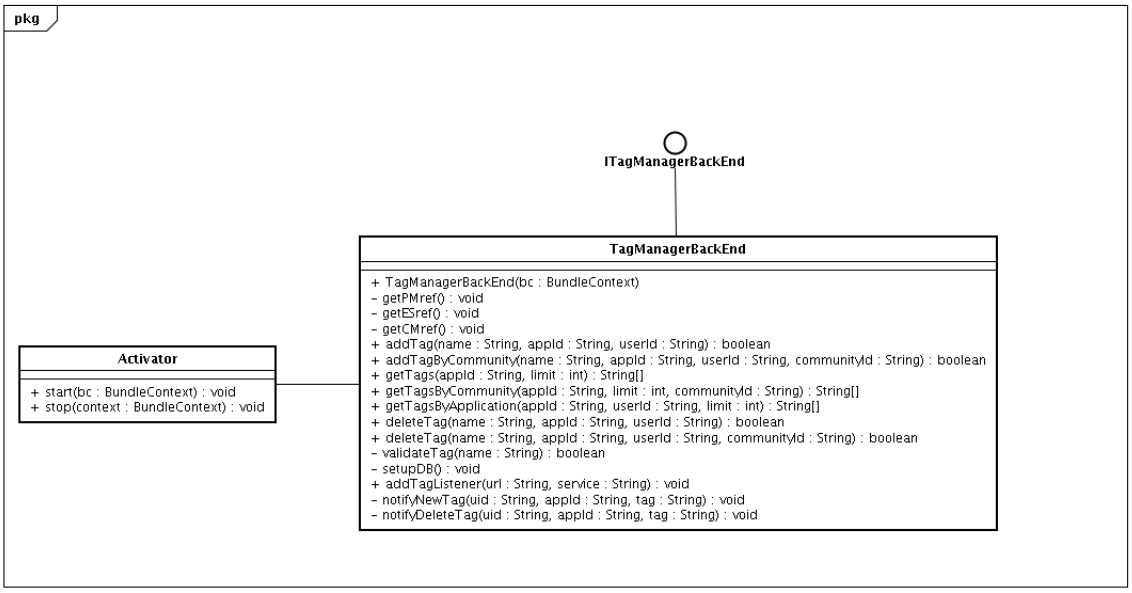

pkg
ITagManagerBackEnd
TagManagerBackEnd
+ TagManagerBackEnd(bc : BundleContext)
- getPMref() : void
- getESref() : void
- getCMref() : void
+ addTag(name : String, appId : String, userId : String) : boolean
+ addTagByCommunity(name : String, appId : String, userId : String, communityId : String) : boolean
+ getTags(appId : String, limit : int) : String[]
+ getTagsByCommunity(appId : String, limit : int, communityId : String) : String[]
+ getTagsByApplication(appId : String, userId : String, limit : int) : String[]
+ deleteTag(name : String, appId : String, userId : String) : boolean
+ deleteTag(name : String, appId : String, userId : String, communityId : String) : boolean
- validateTag(name : String) : boolean
- setupDB() : void
+ addTagListener(url : String, service : String) : void
- notifyNewTag(uid : String, appId : String, tag : String) : void
- notifyDeleteTag(uid : String, appId : String, tag : String) : void
Activator
+ start(bc : BundleContext) : void
+ stop(context : BundleContext) : void

Figure 4.10: TagManagerBackEnd (class diagram)

#### 4.3.3.4 Storage system

The storage system in this case is also simpler: it consists only of one table as in the code in which was based, allowing back compatibility.
The relationships between this table and the repository tables are shown in the previously explained Figure 4.8.

### 4.3.4 TagManagerNode

As it was introduced in Section 4.3.3, `TagManagerNode` is the bundle responsible for the private tags. It is executed in the nodes (client side), and it has an interface to offer its services to the rest of the bundles in ASTRA.

In this section we will explain the process we followed to carry on its design, taking a similar approach to the one we took previously. `TagManagerNode` is similar to `TagManagerBackEnd` and, since it has to take care only of private tags, its design is simpler. Therefore the explanation in this case will not be so detailed and the reader will be forwarded to previous subsections when a similar concept arises.

#### 4.3.4.1 Connection with other bundles

As is shown in Figure 4.11, `TagManagerNode` needs only the services of `PersistencyManager`, which allow it to store its data permanently.

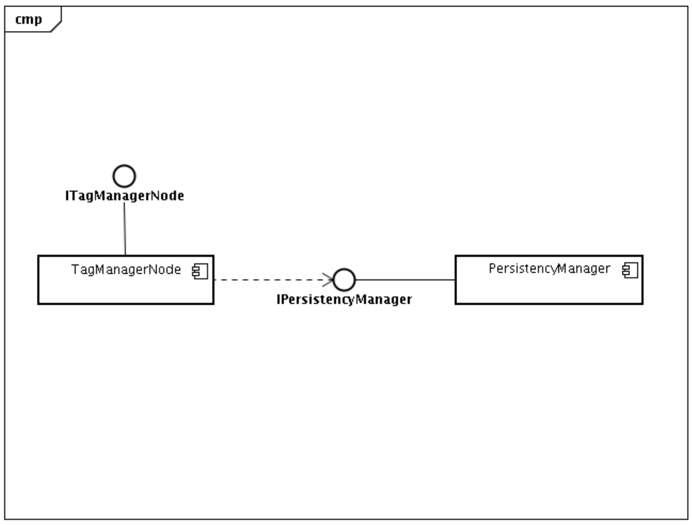


Figure 4.11: TagManagerNode (components diagram)

#### 4.3.4.2 Interface definition

`TagManagerNode`‘s interface is summarize in Table 4.19.

| **addTag** | | |
|---|---|---|
| Description: Creates a new private tag. | | |
| Parameter | Type | Description |
| name | String | Tag name. |
| appId | String | Application identifier in the standard ASTRA format. |
| userId | String | Identifier of the user who tags the application. |
| Returns: Boolean, true if the operation was successful, false otherwise. | | |
| **getTags** | | |
| Description: Returns a list of private tags for the given application. | | |
| Parameter | Type | Description |
| appId | String | Application identifier in the standard ASTRA format. |
| userId | String | Identifier of the user. |
| limit | Integer | Limit on number of results returned. |
| Returns: String array, list of private tags for the given application. | | |
| **deleteTag** | | |
| Description: Deletes a private tag. | | |
| Parameter | Type | Description |
| name | String | Tag name. |
| appId | String | Application identifier in the standard ASTRA format. |
| userId | String | Identifier of the user who tagged the application. |
| Returns: Boolean, true if the operation was successful, false otherwise. | | |

Table 4.19: TagManagerNode interface

#### 4.3.4.3 Classes definition

The main functionalities are provided by the class `TagManagerNode` as is shown in the Figure 4.12.

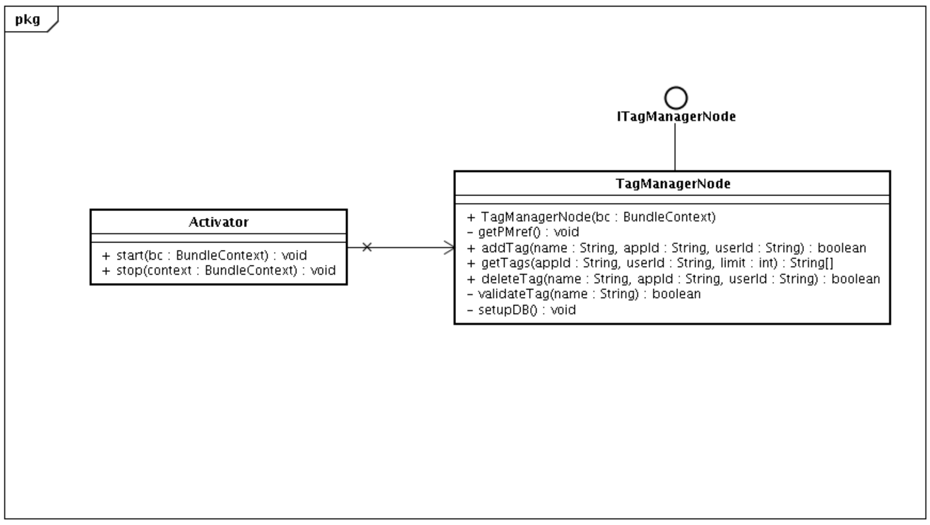

pkg
ITagManagerNode
TagManagerNode
+ TagManagerNode(bc : BundleContext)
- getPMref() : void
+ addTag(name : String, appId : String, userId : String) : boolean
+ getTags(appId : String, userId : String, limit : int) : String[]
+ deleteTag(name : String, appId : String, userId : String) : boolean
- validateTag(name : String) : boolean
- setupDB() : void
Activator
+ start(bc : BundleContext) : void
+ stop(context : BundleContext) : void

Figure 4.12: TagManagerNode (class diagram)

#### 4.3.4.4 Storage system

It consists only of one table, following an schema similar to the one explained in Section 4.3.3.4.

### 4.3.5 ApplicationManager

The last bundle is `ApplicationManager`, which is in charge of the interaction with the user by offering an intuitive GUI, and the connection with other ASTRA bundles services based on that interaction. It is executed in the Node side, and it makes uses of the services offered by the bundles in both edges: Node (client side) and Backend (server side).

In this section we will explain the process we followed to carry on its design. The most important implementation details about this bundle are explained in Section 4.4.

#### 4.3.5.1 Connection with other bundles

First we need to analyze the relationship between this bundle and the rest of bundles in ASTRA. Taking into account the requirements stated in Section 4.1 and the analysis performed in Section 4.2, it was necessary to use the services of the following bundles:

- Local bundles:
  - `AwarenessApplicationManager`: Necessary to send and receive information about the local applications.
  - `AwarenessManager`: Necessary to send and receive information about the local rules.
  - `TagManagerNode`: Necessary to manage the private tags.
  - `OntologyManager`: Necessary to assist the user in the application adaptation process using ontologies[4].
- Remote bundles:
  - `UserManager`: Necessary to authenticate the users during the logging in process.
  - `CommunityManager`: Necessary to retrieve information about the communities joined by the user.
  - `TagManagerBackend`: Necessary to manage the public and community tags.
  - `RepositoryManager`: Necessary to share and retrieve applications with the rest of the users.

The Figure 4.13 shows graphically the relationship between `ApplicationManager` and the rest of bundles.

[4]This feature is not available in the current version of ASTRA, but `ApplicationManager` has been prepared to be easily extended. A detailed explanation can be found in Section 4.4.4.

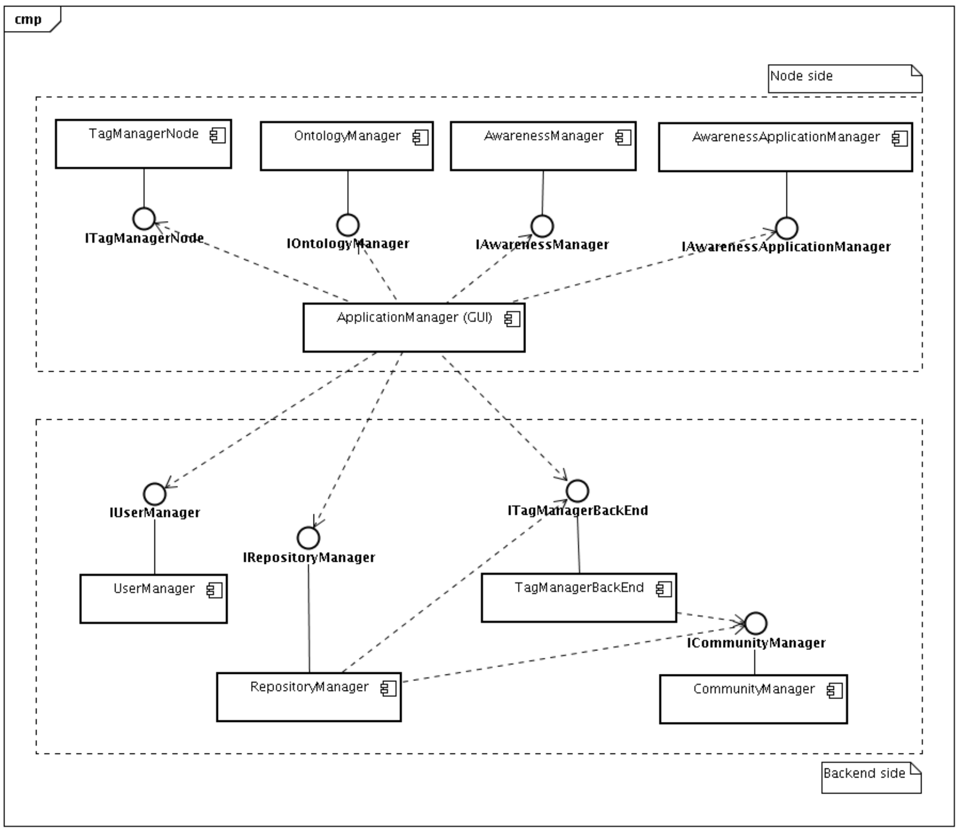


Figure 4.13: ApplicationManager (components diagram)

#### 4.3.5.2 Bundle architecture

In this subsection we will explain briefly the architectural and design patterns which are implemented by this bundle[5], and we will discuss the reasons why they were chosen.

**Singleton (design pattern):**

A design pattern is a description for a commonly recurring structure of communicating components that solves a general design problem within a particular context. It provides a scheme for refining the components of a software system and the relationships between them.
In order to implement the MVC architectural pattern, we made use of the singleton

[5]The implementation details will be stated in Section 4.4.

design pattern in the controller. The singleton pattern makes the class itself responsible for keeping track of its sole instance, ensuring that no other instance can be created.
The Figure 4.14 shows a singleton general class diagram.

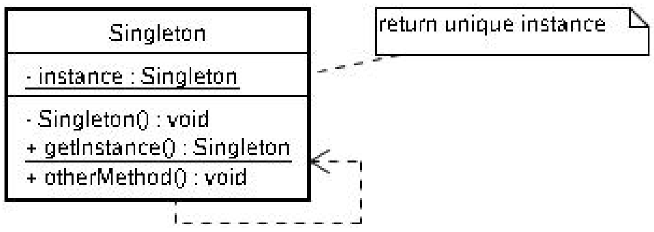


Figure 4.14: Singleton design pattern

**MVC (architectural pattern):**

An architectural pattern expresses a fundamental structural organisation or schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organising the relationships between them.
`ApplicationManager` implements a MVC (Model-View-Controller) pattern, which allows us to isolate the business logic from the user interface, permitting one to be freely modified without affecting the other. The controller collects user input, the model manipulates application data, and the view presents results to the user.
The details about how MVC was implemented in the bundle can be found in Section 4.4, and the design of the classes is explained in Section 4.3.5.3.

#### 4.3.5.3 Classes definition

As we stated in Section 4.3.5.2, `ApplicationManager` implements a MVC pattern. Therefore the class definition is based on the structural organization defined for this pattern which we can summarize as follows:

- Controller: It is represented by the class `ApplicationManagerController`, whose job is to contain the interactions of the user interface components and to interact with other "business logic" classes. It implements a singleton design pattern. It needs basically to perform two functions:

  - To keep track of the references to the user interface components.
  - To provide a set of methods that other components can directly call in their event handler.

- Model: It is represented by the class `ApplicationManagerModel`, which contains the "business logic". Taking into account the bundle nature of `ApplicationManager`, the "business logic" in this case is simpler, summarizing its functions in:

  - To manage the references to the rest of the bundles.
  - To work as an "stub container" to make use of the services provided by those.
  - To take care of the session data, i.e.: the user identifier.

  Considering we are only go to need a single instance of the model, it also implements a singleton pattern[6].

- View: It is represented by several classes whose task consist of presenting the information to the user. This includes all the classes which represent the windows (i.e.: `MainWindow`, `LoginWindow`, etc.) and all the necessary classes that extend some of the graphical components (i.e.: `TreeRenderer`, `TagListCellRenderer`, etc.)[7].

The Figure 4.15 shows the most important classes that made up this bundle and the relationships between them.

#### 4.3.5.4 GUI design

Finally, we need to decide the composition of the graphical user interface. It was divided into the following windows:

[6]This is an specific decission for this bundle, and it is not established by the MVC architectural pattern, actually in most of the cases the model does not implement it.

[7]These classes are ommited in Figure 4.15 in order to make it simpler.

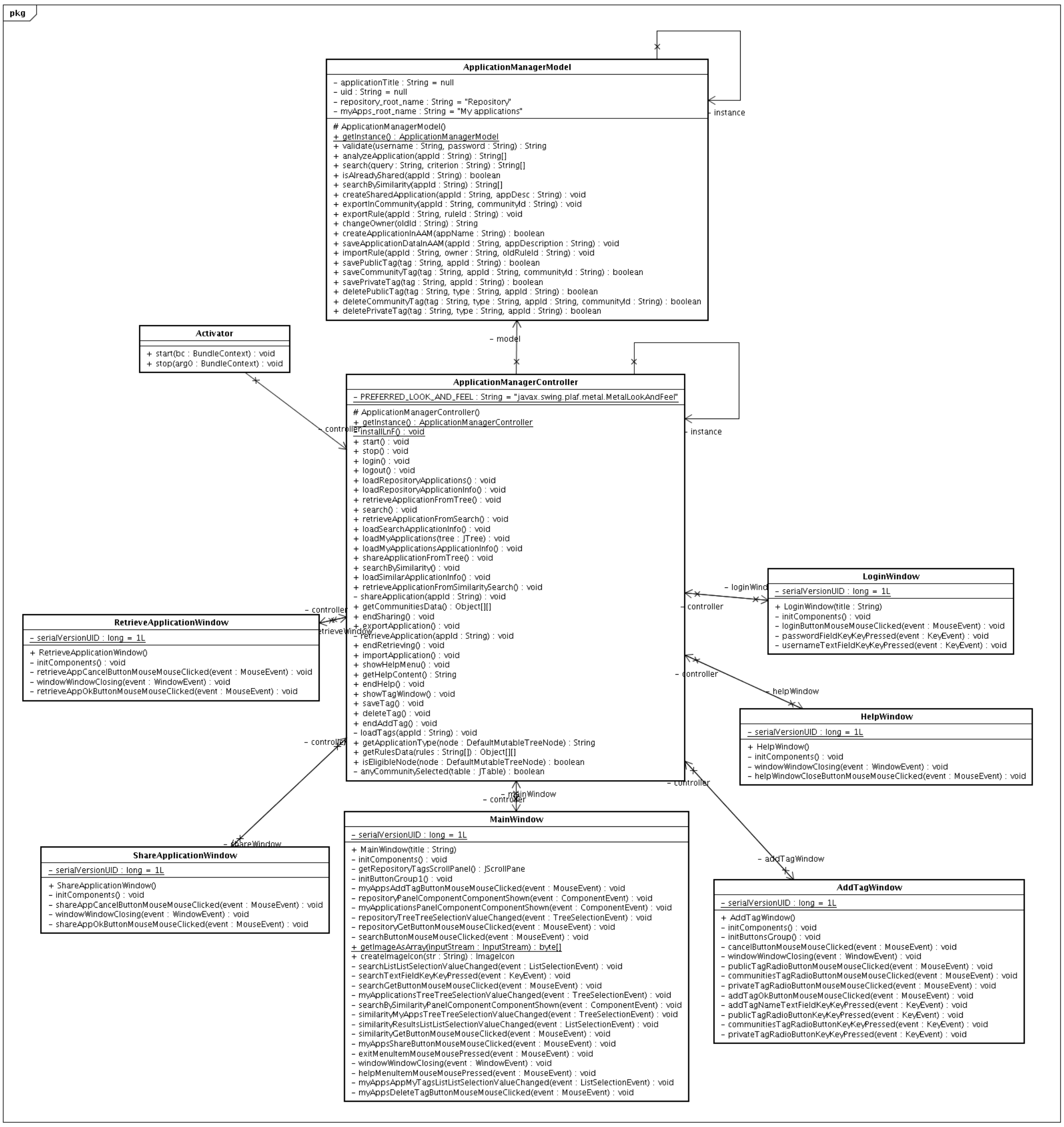

pkg
ApplicationManagerModel
- applicationTitle : String = null
- uid : String = null
- repository_root_name : String = "Repository"
- myApps_root_name : String = "My applications"
# ApplicationManagerModel()
+ getInstance() : ApplicationManagerModel
+ validate(username : String, password : String) : String
+ analyzeApplication(appId : String) : String[]
+ search(query : String, criterion : String) : String[]
+ isAlreadyShared(appId : String) : boolean
+ searchBySimilarity(appId : String) : String[]
+ createSharedApplication(appId : String, appDesc : String) : void
+ exportInCommunity(appId : String, communityId : String) : void
+ exportRule(appId : String, ruleId : String) : void
+ changeOwner(oldId : String) : String
+ createApplicationInAAM(appName : String) : boolean
+ saveApplicationDataInAAM(appId : String, appDescription : String) : void
+ importRule(appId : String, owner : String, oldRuleId : String) : void
+ savePublicTag(tag : String, appId : String) : boolean
+ saveCommunityTag(tag : String, appId : String, communityId : String) : boolean
+ savePrivateTag(tag : String, appId : String) : boolean
+ deletePublicTag(tag : String, type : String, appId : String) : boolean
+ deleteCommunityTag(tag : String, type : String, appId : String, communityId : String) : boolean
+ deletePrivateTag(tag : String, type : String, appId : String) : boolean
- instance
- model
Activator
+ start(bc : BundleContext) : void
+ stop(arg0 : BundleContext) : void
ApplicationManagerController
- PREFERRED_LOOK_AND_FEEL : String = "javax.swing.plaf.metal.MetalLookAndFeel"
# ApplicationManagerController()
+ getInstance() : ApplicationManagerController
- installLnF() : void
+ start() : void
+ stop() : void
+ login() : void
+ logout() : void
+ loadRepositoryApplications() : void
+ loadRepositoryApplicationInfo() : void
+ retrieveApplicationFromTree() : void
+ search() : void
+ retrieveApplicationFromSearch() : void
+ loadSearchApplicationInfo() : void
+ loadMyApplications(tree : JTree) : void
+ loadMyApplicationsApplicationInfo() : void
+ shareApplicationFromTree() : void
+ searchBySimilarity() : void
+ loadSimilarApplicationInfo() : void
+ retrieveApplicationFromSimilaritySearch() : void
- shareApplication(appId : String) : void
+ getCommunitiesData() : Object[][]
+ endSharing() : void
+ exportApplication() : void
- retrieveApplication(appId : String) : void
+ endRetrieving() : void
+ importApplication() : void
+ showHelpMenu() : void
+ getHelpContent() : String
+ endHelp() : void
+ showTagWindow() : void
+ saveTag() : void
+ deleteTag() : void
+ endAddTag() : void
- loadTags(appId : String) : void
+ getApplicationType(node : DefaultMutableTreeNode) : String
+ getRulesData(rules : String[]) : Object[][]
+ isEligibleNode(node : DefaultMutableTreeNode) : boolean
- anyCommunitySelected(table : JTable) : boolean
- instance
- controller
- retrieveWindow
- loginWindow
- helpWindow
- mainWindow
- shareWindow
- addTagWindow
LoginWindow
- serialVersionUID : long = 1L
+ LoginWindow(title : String)
- initComponents() : void
- loginButtonMouseMouseClicked(event : MouseEvent) : void
- passwordFieldKeyKeyPressed(event : KeyEvent) : void
- usernameTextFieldKeyKeyPressed(event : KeyEvent) : void
RetrieveApplicationWindow
- serialVersionUID : long = 1L
+ RetrieveApplicationWindow()
- initComponents() : void
- retrieveAppCancelButtonMouseMouseClicked(event : MouseEvent) : void
- windowWindowClosing(event : WindowEvent) : void
- retrieveAppOkButtonMouseMouseClicked(event : MouseEvent) : void
HelpWindow
- serialVersionUID : long = 1L
+ HelpWindow()
- initComponents() : void
- windowWindowClosing(event : WindowEvent) : void
- helpWindowCloseButtonMouseMouseClicked(event : MouseEvent) : void
MainWindow
- serialVersionUID : long = 1L
+ MainWindow(title : String)
- initComponents() : void
- getRepositoryTagsScrollPanel() : JScrollPane
- initButtonGroup1() : void
- myAppsAddTagButtonMouseMouseClicked(event : MouseEvent) : void
- repositoryPanelComponentComponentShown(event : ComponentEvent) : void
- myApplicationsPanelComponentComponentShown(event : ComponentEvent) : void
- repositoryTreeTreeSelectionValueChanged(event : TreeSelectionEvent) : void
- repositoryGetButtonMouseMouseClicked(event : MouseEvent) : void
- searchButtonMouseMouseClicked(event : MouseEvent) : void
+ getImageAsArray(inputStream : InputStream) : byte[]
+ createImageIcon(str : String) : ImageIcon
- searchListListSelectionValueChanged(event : ListSelectionEvent) : void
- searchTextFieldKeyKeyPressed(event : KeyEvent) : void
- searchGetButtonMouseMouseClicked(event : MouseEvent) : void
- myApplicationsTreeTreeSelectionValueChanged(event : TreeSelectionEvent) : void
- searchBySimilarityPanelComponentComponentShown(event : ComponentEvent) : void
- similarityMyAppsTreeTreeSelectionValueChanged(event : TreeSelectionEvent) : void
- similarityResultsListListSelectionValueChanged(event : ListSelectionEvent) : void
- similarityGetButtonMouseMouseClicked(event : MouseEvent) : void
- myAppsShareButtonMouseMouseClicked(event : MouseEvent) : void
- exitMenuItemMouseMousePressed(event : MouseEvent) : void
- windowWindowClosing(event : WindowEvent) : void
- helpMenuItemMouseMousePressed(event : MouseEvent) : void
- myAppsAppMyTagsListListSelectionValueChanged(event : ListSelectionEvent) : void
- myAppsDeleteTagButtonMouseMouseClicked(event : MouseEvent) : void
ShareApplicationWindow
- serialVersionUID : long = 1L
+ ShareApplicationWindow()
- initComponents() : void
- shareAppCancelButtonMouseMouseClicked(event : MouseEvent) : void
- windowWindowClosing(event : WindowEvent) : void
- shareAppOkButtonMouseMouseClicked(event : MouseEvent) : void
AddTagWindow
- serialVersionUID : long = 1L
+ AddTagWindow()
- initComponents() : void
- initButtonsGroup() : void
- cancelButtonMouseMouseClicked(event : MouseEvent) : void
- windowWindowClosing(event : WindowEvent) : void
- publicTagRadioButtonMouseMouseClicked(event : MouseEvent) : void
- communitiesTagRadioButtonMouseMouseClicked(event : MouseEvent) : void
- privateTagRadioButtonMouseMouseClicked(event : MouseEvent) : void
- addTagOkButtonMouseMouseClicked(event : MouseEvent) : void
- addTagNameTextFieldKeyKeyPressed(event : KeyEvent) : void
- publicTagRadioButtonKeyKeyPressed(event : KeyEvent) : void
- communitiesTagRadioButtonKeyKeyPressed(event : KeyEvent) : void
- privateTagRadioButtonKeyKeyPressed(event : KeyEvent) : void

Figure 4.15: ApplicationManager (class diagram)

- `LoginWindow`: Displays a window where the user can introduce his user name and password.
- `MainWindow`: Displays a window where the user can manage the local and

remote applications. It is divided into the following tabs:

- "My applications":
    - It allows the user to browse his applications, see the information about them and share them in the repository.
    - It allows the user to manage tags associated to certain application.
- "Repository": It allows the user to browse the repository, to see information about the applications and to retrieve them.
- "Search": It allows the user to perform queries to look for applications in the repository by different criteria: description, tags, type or any.
- "Search by similarity": It allows the user to search applications by choosing one he has in his local space.

It will also offer a menu tab with options to look up for remote help and to log out of the system.

- `ShareApplicationWindow`: Displays a window where the user can customize the parameters of the sharing process: select the communities, select the rules, visualize their description, etc.
- `RetrieveApplicationWindow`: Displays a window where the user can customize the parameters of the retrieving process: select the rules, visualize their description, change the application description, etc.
- `AddTagWindow`: Displays a window where the user can tag an application and define its visibility.
- `HelpWindow`: Displays a window with the help information.

The user interaction through this GUI is detailed in Section 4.5.

## 4.4 Implementation

In this section we will describe some of the most important implementation details that were undertaken during the development phase.

### 4.4.1 Creating and deploying a bundle

In this section we will enumerate the necessary steps to create an OSGi bundle and deploy it in the Knopflerfish framework, assuming we have already set up the necessary tools (Eclipse, Knopflerfish eclipse plugin, etc) [8]. It is based on the tutorials in [Hai04] and [Bar09].

#### 4.4.1.1 Creating the project and the manifest

First of all, we need to create an OSGi project and ensure that the `framework.jar` file is imported in the Java Build Path, otherwise we will not be able to access the OSGi classes and interfaces provided by Knopflerfish.
One of the differences with respect to a common Java project is the need of defining a manifest file which contains the description of the bundle. This manifest is used by the framework to get information about it and to deploy it successfully. As an example we show a simplified version of `RepositoryManager`‘s manifest:

```
Manifest-Version: 1.0
Export-Package: eu.ist.astra.rm,eu.ist.astra.rm.impl
Bundle-Vendor: IST ASTRA
Bundle-ClassPath: ., lib/lucene-2_4_1_svn.jar
Bundle-Version: 2.0.0
Bundle-Name: RepositoryManager
Bundle-Activator: eu.ist.astra.rm.impl.Activator
Bundle-Description: Repository Manager bundle which manages the sharing
process in the back-end
Bundle-SymbolicName: RepositoryManager
Import-Package: org.osgi.framework, eu.ist.astra.persistency,
eu.ist.astra.cm, eu.ist.astra.tmbe
(...)
```

[8]Details about the tools can be found in Section 4.8.

From which we can remark the following attributes:

- Export-Packages: It informs the framework which are the classes and interfaces offered by this bundle.
- Bundle-Classpath: It is necessary to establish all the external libraries used by this bundle.
- Bundle-Activator: It tells the framework which class is the `Activator` class, it is a similar concept to the "main" class in a normal Java application.
- Import-Package: It informs the framework about the bundles it needs to have access to (only locals).

Appendix C contains all the manifests from all the bundles developed for this project.
It is also necessary to create an Ant Build file to build the project, but assuming we are using OSGi Knopflefish plugin for Eclipse, it is created automatically.

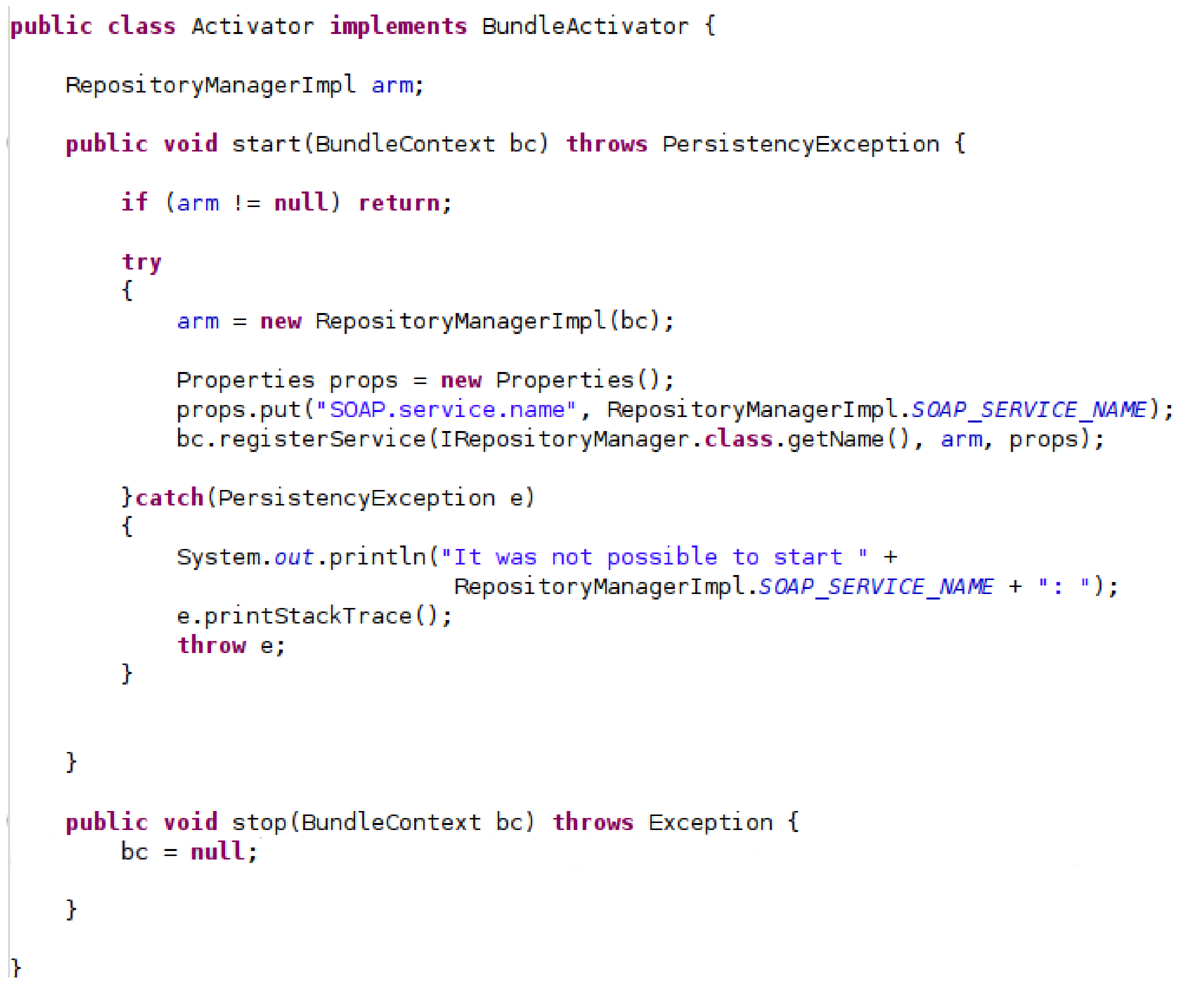
#### 4.4.1.2 Creating an activator

The next step consists of defining the `Activator` class, a class that implements the `BundleActivator` interface. This interface requires the implementation of two methods, `start()` and `stop()`, which are used by the framework to manage the bundle. The `start()` and `stop()` methods receive a `BundleContext` object. We have to store this object once we get it and set the reference back to `null` when the bundle is stopped. That way, the Garbage Collector can do its work and free unused resources. We have also to register the services by calling the `registerService()` method of the `BundleContext`. It receives three parameters: the first parameter is the name of the service interface. The second is the service implementation. The third parameter can be used to supply additional information about the service as key/value pairs.

As an example, Figure 4.16 shows `RepositoryManager`'s activator code.

#### 4.4.1.3 Running the bundles

Finally, we just need to configure the running configuration. An easy way to do it is by creating a OSGi run configuration, where we can set the framework, configure the system properties, etc. The most delicate part consists of defining the priorities to run



```
public class Activator implements BundleActivator {

    RepositoryManagerImpl arm;

    public void start(BundleContext bc) throws PersistencyException {

        if (arm != null) return;

        try
        {
            arm = new RepositoryManagerImpl(bc);

            Properties props = new Properties();
            props.put("SOAP.service.name", RepositoryManagerImpl.SOAP_SERVICE_NAME);
            bc.registerService(IRepositoryManager.class.getName(), arm, props);

        }catch(PersistencyException e)
        {
            System.out.println("It was not possible to start " +
                                    RepositoryManagerImpl.SOAP_SERVICE_NAME + ": ");
            e.printStackTrace();
            throw e;
        }


    }

    public void stop(BundleContext bc) throws Exception {
        bc = null;

    }

}
```

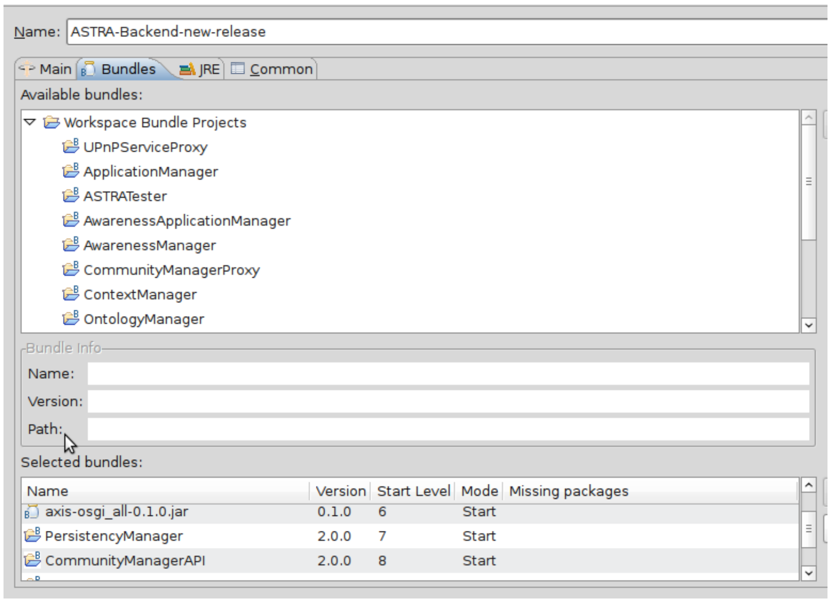
Figure 4.16: RepositoryManager's activator

the bundles. This can be problematic for those bundles that make use of other bundles services when they are run for the first time[9], i.e: the need of `RepositoryManager` to access the database using `PersistencyManager`'s services when is started for the first time.

The Figure 4.17 shows a screenshot where the bundles priorities are configured using the OSGi Eclipse plugin. The configurations for running an ASTRA Node and the ASTRA Backend are explained in Appendix D.

After all these steps our bundle can be successfully run and debugged.

[9]An alternative consists of creating a `ServiceTracker` object, but this is out of the scope of this explanation.



Name: ASTRA-Backend-new-release

Available bundles:

- Workspace Bundle Projects
  - UPnPServiceProxy
  - ApplicationManager
  - ASTRATester
  - AwarenessApplicationManager
  - AwarenessManager
  - CommunityManagerProxy
  - ContextManager
  - OntologyManager

Bundle Info

Name:

Version:

Path:

Selected bundles:

| Name | Version | Start Level | Mode | Missing packages |
|---|---|---|---|---|
| axis-osgi_all-0.1.0.jar | 0.1.0 | 6 | Start | |
| PersistencyManager | 2.0.0 | 7 | Start | |
| CommunityManagerAPI | 2.0.0 | 8 | Start | |

Figure 4.17: Bundles priorities configuration

## 4.4.2 Implementing MVC in a SWING application

In this section we will explain the process carried on to develop the `ApplicationManager` GUI (whose design was discussed in Section 4.3.5) following a MVC architectural pattern.

It is based in some of the ideas explained in [Lyb07] and [GM05].

We will summarize it describing the responsibilities of each sub-component:

- Controller: It is implemented in a singleton class, which
  - Keeps track of the user interface components.
  - Provides a set of methods that can be called by the events handlers.
- View: All the windows were designed graphically, using a free software plugin for Eclipse called "Visual Swing 4 Eclipse"[10]. It offers also facilities to create "empty" methods that handle the events. This makes the development faster since we only have to:
  - Set the references to the components in the controller.

[10]More details can be found in Section 4.8.

  - Establish a relationship between the proper event and the method to be called in the controller.

- Model: Implements the "business logic" that, taking into account the SOA nature of the whole project, consists of:

  - Store the user session information and the references to the rest of the bundles.
  - Provides a set of methods with the proper information to be consumed by the controller, and make use of the services offered by other bundles when necessary.

As it was briefly discussed in Section 4.3.5, we assure a loose coupling relationship between the subcomponents thanks to it.

It is also interesting to remark that the GUI creates background threads to perform time-consuming tasks, by extending the `SwingWorker` class in order to keep the GUI responsive [Sun08]. Concretely, threads are created for loading the tree hierarchy for remote and local applications.

### 4.4.3 Search engine development

In order to accomplish the requirements related to searching capabilities (see Section 4.1) which are captured in Section 4.2.2, we decided to use the open source library Lucene (see 3.2.7).
Lucene allowed us to provide search capabilities in a really flexible and powerful way. In this section we will explain the steps we took to create and integrate the search engine based on this library into `RepositoryManager` (see Section 4.3.2).

#### 4.4.3.1 Creating the index

The first step consisted of deciding how the index is going to be created.
Lucene stores the index in a directory which can be of two different types: `FSDirectory`, storing the contents in the contents in the file system, or `RAMDirectory`, storing the contents in memory. Our search engine uses the latter, since we do not need persistence of this index once `RepositoryManager` is stopped, and it offers a better performance.

Lucene defines a `Document` as the basic unit to be indexed. Every document is composed of `Fields`, which represent all the information associated to the `Document`. In our case the notion of what a `Document` is seems straight: an application, but deciding what information to store about it (the `Fields`) requires more caution. Lucene allows to customize the type of field by configuring how the field is going to be stored (see Table 4.20) and how is going to be indexed (see Table 4.21).

| **Type of storage** | |
|---|---|
| Type | Description |
| Compress | Store the original field value in the index in a compressed form. |
| Yes | Do not store the field value in the index. |
| No | Store the original field value in the index. |

Table 4.20: Types of field storage options in Lucene

| **Type of indexation** | |
|---|---|
| Type | Description |
| No | Do not index the field value. |
| Analyzed | Index the tokens produced by running the field's value through an Analyzer. |
| Not Analyzed | Index the field's value without using an Analyzer, so it can be searched. |

Table 4.21: Types of field indexation in Lucene

Taking into account these field types, the Table 4.22 summarizes the type of fields we decided to use and a brief description with the reasons.

The reason why we need to store all the fields is based on the lack of functionalities to update the document fields in the index. Our search engine needs to take into account the addition and removal of some fields associated to a document dynamically (i.e.: the tags which are added or removed), therefore we need to update that document in the index. The only way to implement this in the current version of Lucene is by retrieving and deleting the old version of the document, and creating and indexing the new version of the document (using information from the old one if it is necessary).

| **Search engine - fields** | | | |
|---|---|---|---|
| Field | Storage | Indexation | Description |
| App. identifier | Yes | Not Analyzed | Necessary to return as a result of a query. |
| App. description | Yes | Analyzed | Used for querying. |
| App. type | Yes | Not Analyzed | Used for querying. |
| App. owner | Yes | Not Analyzed | Used to filter applications which belong to the user who is performing the query |
| App. tags | Yes | Analyzed | Used for querying. |

Table 4.22: Fields used by RepositoryManager's search engine index

#### 4.4.3.2 Querying

Once we have implemented the index, we have to create a way to perform the queries. Lucene offers us the possibility of querying in one or in many fields instantiating the classes `QueryParser` or `MultiFieldQueryParser` respectively. In the case of searching by criteria, the matching is almost straight as is shown in Table 4.23.

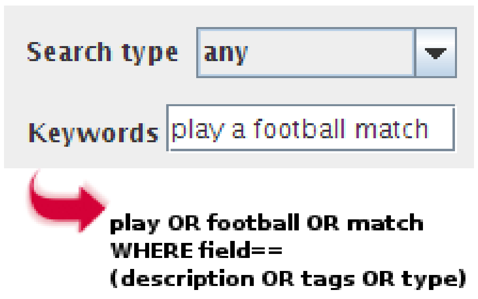
| **Querying by criteria** | | |
|---|---|---|
| Criterion | Type of query parser | Affected field(s) |
| By description | `QueryParser` | App. description |
| By tags | `QueryParser` | App. tags |
| By type | `QueryParser` | App. type |
| Any | `MultiFieldQueryParser` | App. description, tags and type |

Table 4.23: Matching affected fields in querying by criteria

Thank to this mechanism, `RepositoryManager` offers a transparent way to perform the queries easily with the options selected by the user in the GUI. Therefore we can now transform a sentence in natural language into a query in an intuitive way, as is shown in Figure 4.18.

In the case of the search by similarity a deeper analysis was needed, but after several tests, which will be detailed in Section 4.6, the queries are composed by the following data of the application which is going to be compared:

- Application description.



play a football match

play OR football OR match
WHERE field==
(description OR tags OR type)

Figure 4.18: Querying the search engine by using the GUI

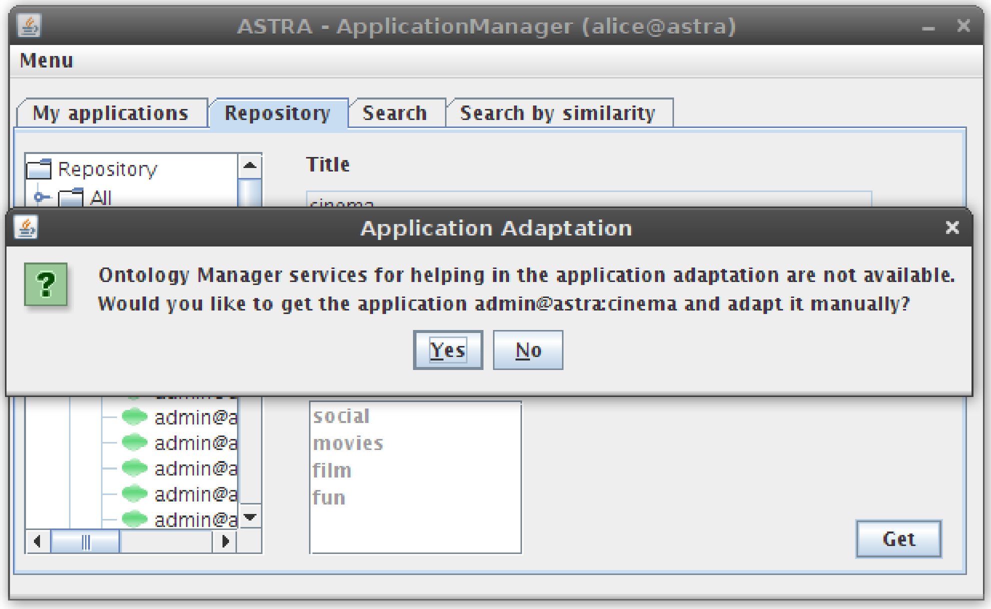
- List of public tags.
- List of community tags associated to that application (which are visible for that user).

The query is composed in a similar way as in the one explained in Figure 4.18, and the search is performed on the fields "description" and "tags" for every application indexed by the search engine.

### 4.4.4 Application adaptation

In this section we will explain the details related to the adaptation of an application retrieved from the repository. As it will be explained in Section 5.3, this process aims to have assistance by inferring using ontologies in the future. The current state of the implementation performs some automatic changes, but it still relies on the user's interaction.
On the other hand, the design and implementation have already taken into account this possible expansion, so they are already connected with `OntologyManager` and call its services (which for the moment return a `null` value) through its interface. Since those methods are not implemented yet, `ApplicationManager` offers the possibility of performing this process manually as is shown in Figure 4.19.

We can divide the current adaptation functionalities offered in two sets: some which are performed directly by the user and some that are performed automatically in a transparent way.
As is shown in Figure 4.20, the user can currently modify the description of the application and select the rules before retrieving the application.

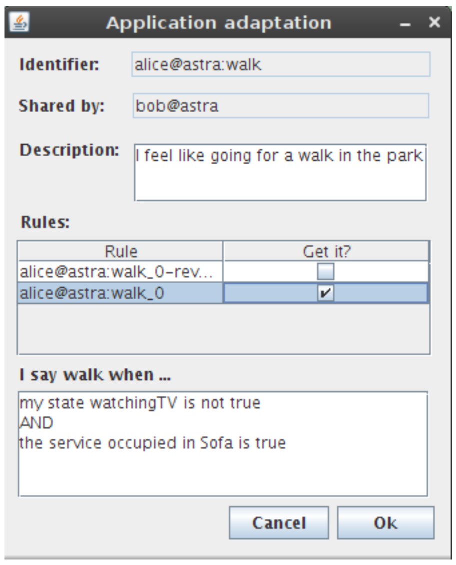


Application Adaptation

Ontology Manager services for helping in the application adaptation are not available.
Would you like to get the application admin@astra:cinema and adapt it manually?

Yes No

Figure 4.19: Offering the alternative of adapting the application manually (screenshot)



Application adaptation

Identifier: alice@astra:walk

Shared by: bob@astra

Description: I feel like going for a walk in the park

Rules:

| Rule | Get it? |
|---|---|
| alice@astra:walk_0-rev... | ☐ |
| alice@astra:walk_0 | ✓ |

I say walk when ...

my state watchingTV is not true
AND
the service occupied in Sofa is true

Cancel Ok

Figure 4.20: Application adaptation through the GUI (screenshot)

The current automatic changes consist of changing the application and rules identifiers (see Figure 4.20) and changing the rules ownership internally. The first one is almost straight, since it is only necessary to reconstruct the Strings which identify

the application or the rule; but the internal changes in the rules require a deeper analysis.
The way in that we can access a rule is through an XML file which describes it[11].

An ASTRA XML rule has the following elements:

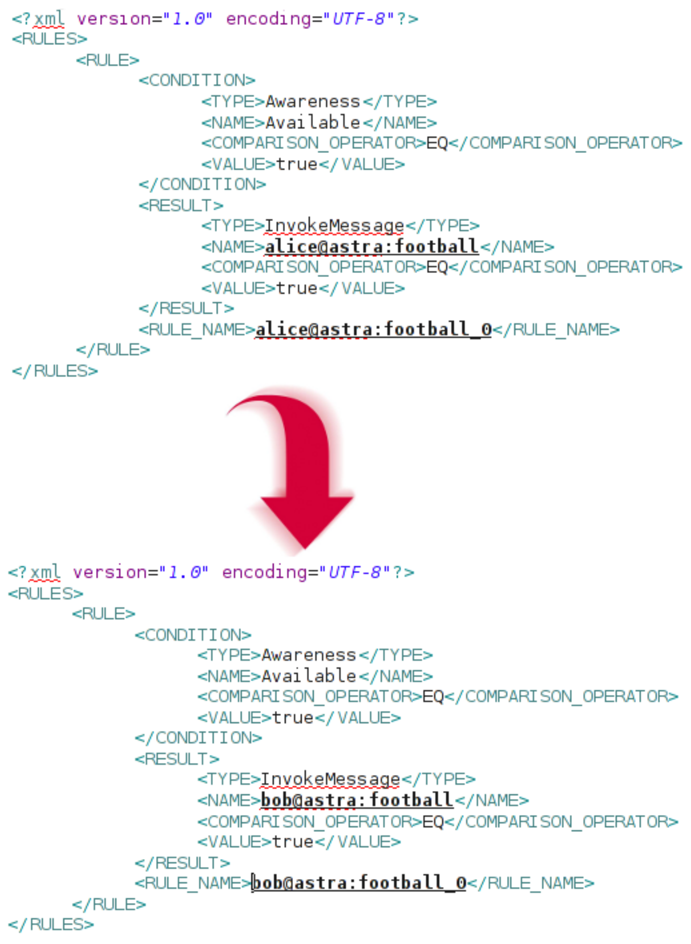
- `RULES`: The main rule tag that contains the rules. Each `RULES` can contain more than one `RULE`.
- `RULE`: The tag that contains a single rule. Each rule needs a `CONDITION` part and a `RESULT` part.
- `RULE_NAME`: The name of the rule.
- `CONDITION`: The "if part" of the rule.
- `TYPE`: The type of condition:
  - If `TYPE` is `AND` or `OR` type then it is followed by `CONDITION_PART`.
  - If the `TYPE` is Service, Awareness or InvokeMessage then it is followed by `NAME`, `OPERATOR` and `VALUE`.
- `CONDITION_PART`: Defines the parts of a complex condition of an `AND` or `OR` type. It always contains two `CONDITION` tags that define the two nested conditions.
- `NAME`: The name of the variable for the condition.
  - If the `TYPE` is Service, the notation is `<device>@<service>` where the `<device>` is the device that has the `<service>`.
  - If the `TYPE` is Awareness, it is an intermediate state that is usually described in the Awareness Ontology.
  - If the `TYPE` is InvokeMessage, it is the name of the application that we want to trigger.
- `OPERATOR`: The operator that describes the relationship between the `NAME` and the `VALUE`. Can be "eq", "neq", "gt", "ge", "lt", "le". It can be omitted, in which case the default value is "eq".

[11]This XML file is transformed to perform inference by a subcomponent called `XMLToClipsParser`, but this is out of the scope of this explanation.

- `VALUE`: The value that is compared with the `NAME` in order to validate the condition.

- `RESULT`: The “then part” of the condition. It is formed in the same way as a condition except that no operator is needed.

The Figure 4.21 shows an example of a simple XML rule shared in the repository by user A (Alice), and the same XML rule after applying the required changes in the ownership once user B (Bob) has retrieved it. In general, this changes affect:



```
<?xml version="1.0" encoding="UTF-8"?>
<RULES>
        <RULE>
                <CONDITION>
                        <TYPE>Awareness</TYPE>
                        <NAME>Available</NAME>
                        <COMPARISON_OPERATOR>EQ</COMPARISON_OPERATOR>
                        <VALUE>true</VALUE>
                </CONDITION>
                <RESULT>
                        <TYPE>InvokeMessage</TYPE>
                        <NAME>alice@astra:football</NAME>
                        <COMPARISON_OPERATOR>EQ</COMPARISON_OPERATOR>
                        <VALUE>true</VALUE>
                </RESULT>
                <RULE_NAME>alice@astra:football_0</RULE_NAME>
        </RULE>
</RULES>
```

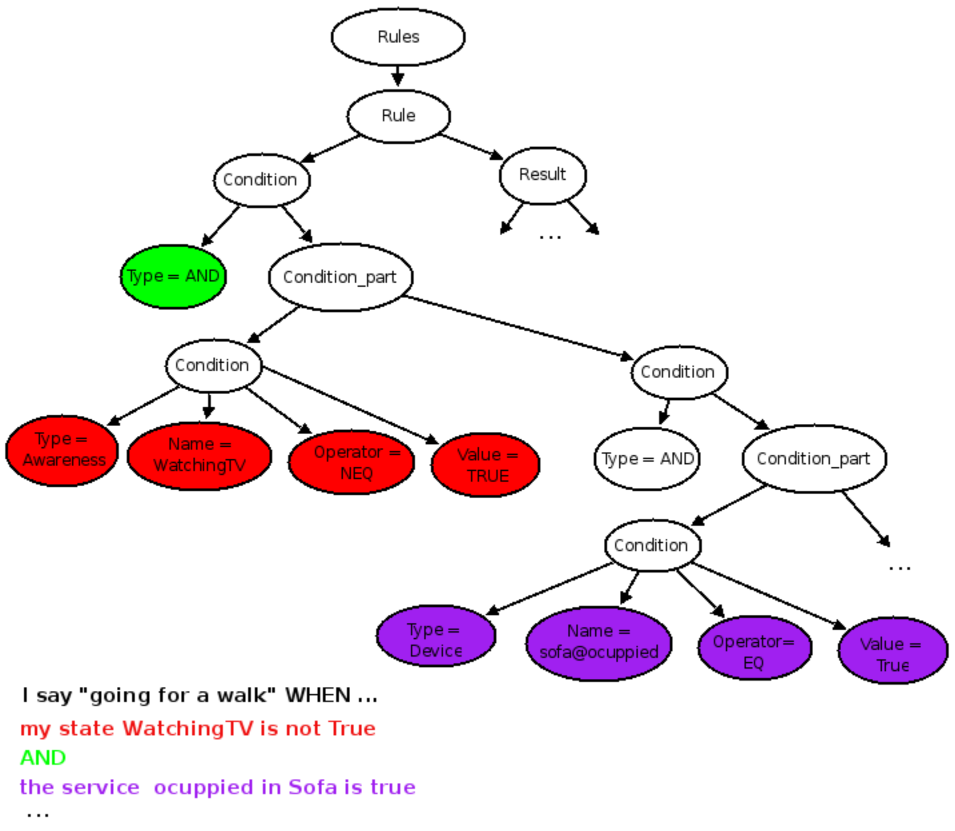
```
<?xml version="1.0" encoding="UTF-8"?>
<RULES>
        <RULE>
                <CONDITION>
                        <TYPE>Awareness</TYPE>
                        <NAME>Available</NAME>
                        <COMPARISON_OPERATOR>EQ</COMPARISON_OPERATOR>
                        <VALUE>true</VALUE>
                </CONDITION>
                <RESULT>
                        <TYPE>InvokeMessage</TYPE>
                        <NAME>bob@astra:football</NAME>
                        <COMPARISON_OPERATOR>EQ</COMPARISON_OPERATOR>
                        <VALUE>true</VALUE>
                </RESULT>
                <RULE_NAME>bob@astra:football_0</RULE_NAME>
        </RULE>
</RULES>
```

Figure 4.21: Changes in the internal ownership of the rule

- All the nodes `NAME` that are children of `RESULT`.

- The node `RULE_NAME`.

A similar approach was taken to create a description of the rule based on the XML file (as the one shown in Figure 4.20) in order to help the user to decide which rules he wants to appropriate. The Figure 4.22 shows graphically the way in which we create a human readable description from a rule while analyzing recursively the tree which represents the XML rule for an application (i.e.: `"bob@astra:walk"`).



I say "going for a walk" WHEN ...
my state WatchingTV is not True
AND
the service ocuppied in Sofa is true
...

Figure 4.22: Creating a rule description from an XML file

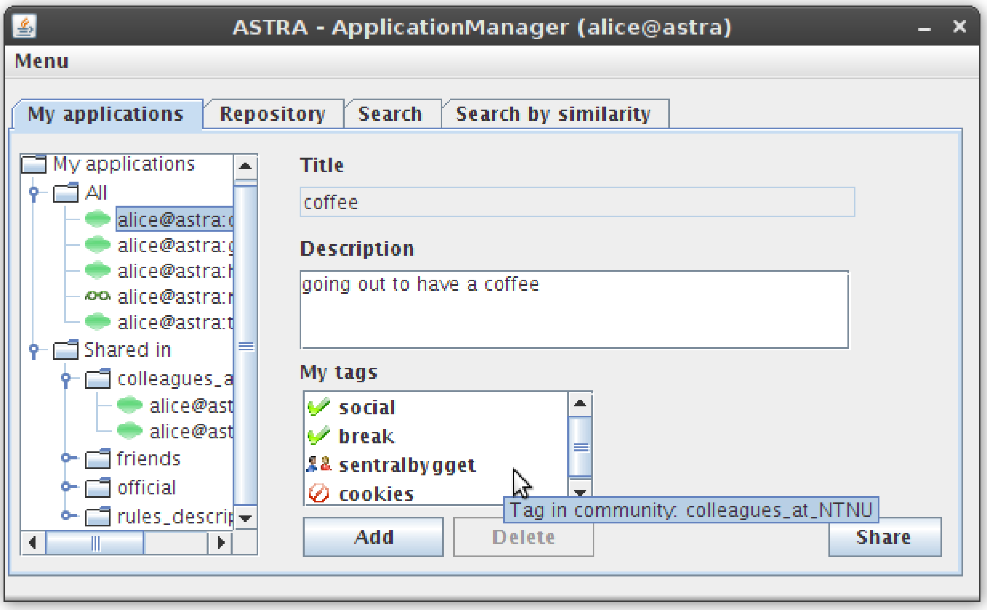
This was implemented using DOM (see Section 3.2.6) in the `RepositoryManager`, and it is accessible for the rest of the bundles by calling the method `getXMLRule()` and `getRuleDescription()` respectively.

## 4.5 User interaction

In this section we will explain some of the main functionalities from the point of view of the user by showing different screenshots. We will focus on 3 actions: "tagging and sharing an application", "locating an application" and "retrieving and adapting and application from the repository".

### 4.5.1 Tagging and sharing

Once the user has successfully logged in the system, he can browse the tree to see all the information related to his applications in the tab "My applications" as is shown in Figure 4.23. In order to distinguish quickly between a focus and a nimbus application (see Section 1.2), the applications are presented with a different icon (cloud for nimbus, glasses for focus) depending on the type.



My applications
- All
  - alice@astra:
  - alice@astra:
  - alice@astra:
  - alice@astra:
  - alice@astra:
- Shared in
  - colleagues_a
    - alice@ast
    - alice@ast
  - friends
  - official
  - rules_descri

Title: coffee

Description: going out to have a coffee

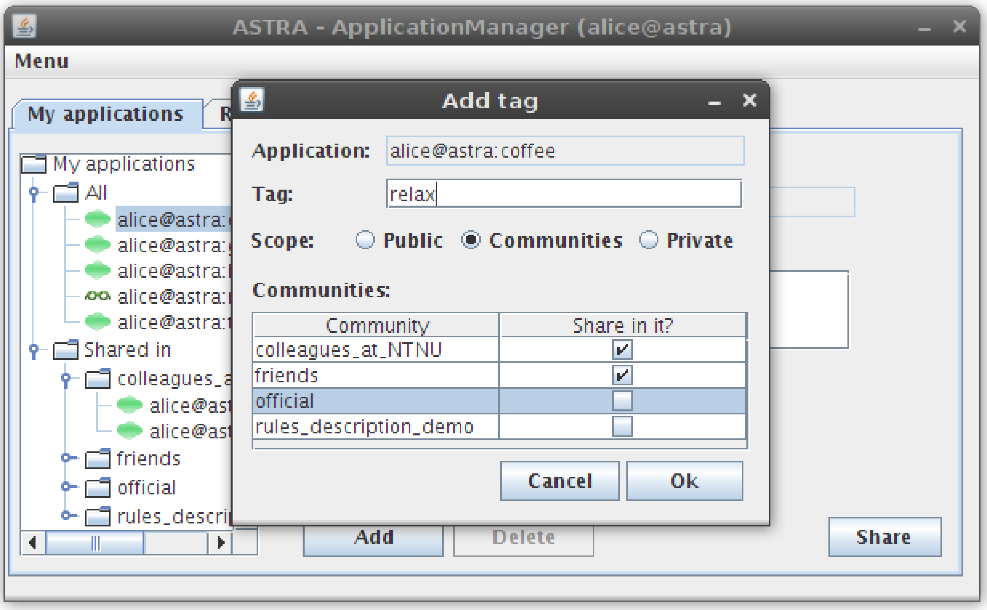
My tags:
- ✓ social
- ✓ break
- sentralbygget
- cookies

Tag in community: colleagues_at_NTNU

Figure 4.23: Interaction with "My applications" (screenshot)

He can manage the tags associated to every application, deleting them or adding new ones. To make the GUI more intuitive, they are represented with a different icon depending on the scope as is shown in Figure 4.23. There is also a tool tip for every of them, where the user can see the scope and the community which belongs to in the case of a community tag.

When the user decides to add a tag, a new window where the user can customize the scope appears, as is shown in Figure 4.24.



Application: alice@astra:coffee

Tag: relax

Scope: ( ) Public [✓ Communities] ( ) Private

Communities:

| Community | Share in it? |
|---|---|
| colleagues_at_NTNU | ✓ |
| friends | ✓ |
| official | |
| rules_description_demo | |

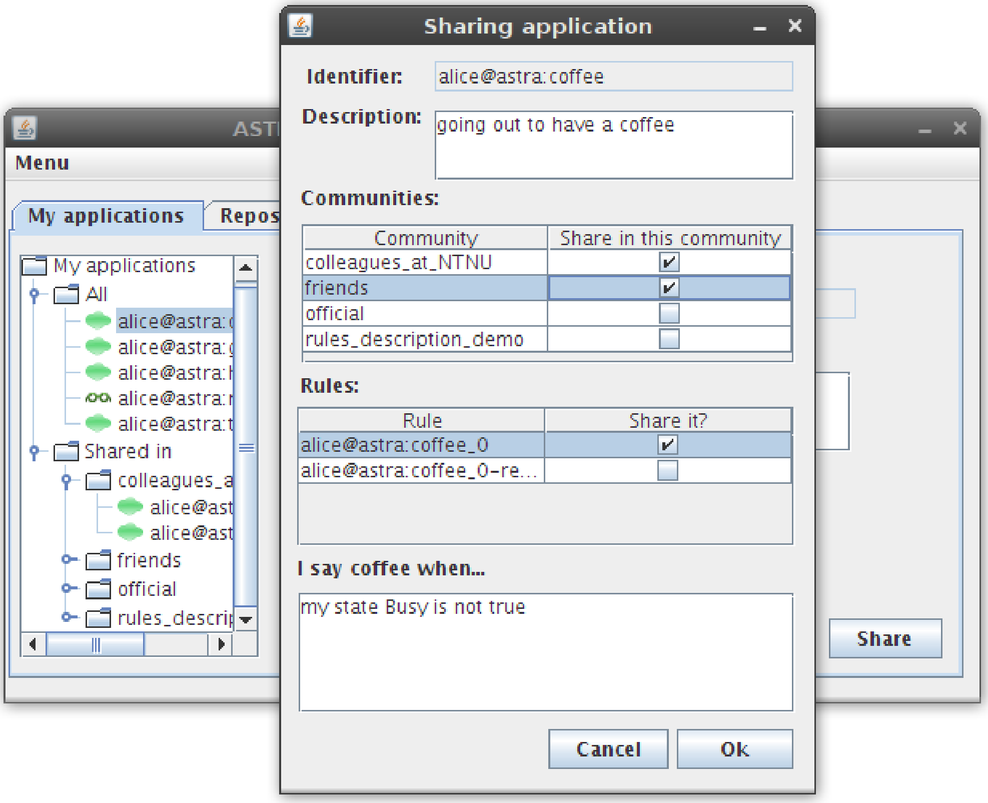
Figure 4.24: Adding a tag (screenshot)

When the user desires to share an application by pressing the button "Share", another window is displayed. Here he can select the communities where the application is going to be visible, the rules he wants to share, change the description, etc. The Figure 4.25 shows an screenshot with that window.

## 4.5.2 Locating an application in the repository

There are three different ways to locate an application in the repository. The first one is offered in the tab "Repository", where the user can browse the repository and see the main information (description, associated visible tags, etc) of the applications which are within his scope. The Figure 4.26 shows the interaction through this tab.

The second one is by performing a query, clicking on the tab "Search". Here the user has to select the criteria (any, by description, by tags or by type), type a query (in natural language, using keywords, etc.) and press the button "Go!". Then the results appear in the list below sorted by score, and the user can see the information related to that results by clicking on them. The Figure 4.27 shows an screenshot of the interaction through this tab.



Identifier: alice@astra:coffee

Description: going out to have a coffee

Communities:

| Community | Share in this community |
|---|---|
| colleagues_at_NTNU | ✓ |
| friends | ✓ |
| official | ☐ |
| rules_description_demo | ☐ |

Rules:

| Rule | Share it? |
|---|---|
| alice@astra:coffee_0 | ✓ |
| alice@astra:coffee_0-re... | ☐ |

I say coffee when...

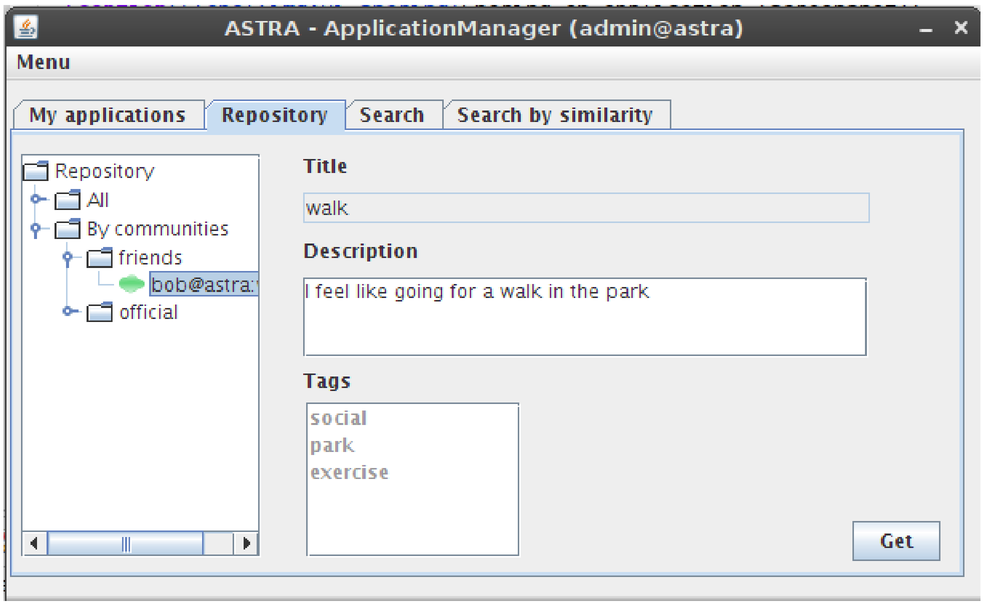
my state Busy is not true

Figure 4.25: Sharing an application (screenshot)

The last option to locate an application is offered in the tab "Search by similarity". Here the user selects one of his applications, and similar applications sorted by similarity are loaded on the list on the right. Selecting one of these applications, the user can see the information related to it. The Figure 4.28 shows the interaction through this tab.

### 4.5.3 Retrieving and adapting an application from the repository

Once the user has located an application by one of the several ways explained in Section 4.5.2 and he decides to get it, a new window where the user can adapt the application is displayed. He can change the description and choose the rules he wants to retrieve. The Figure 4.29 shows an screenshot with the interaction through this window after having performed a search by similarity.



Title

walk

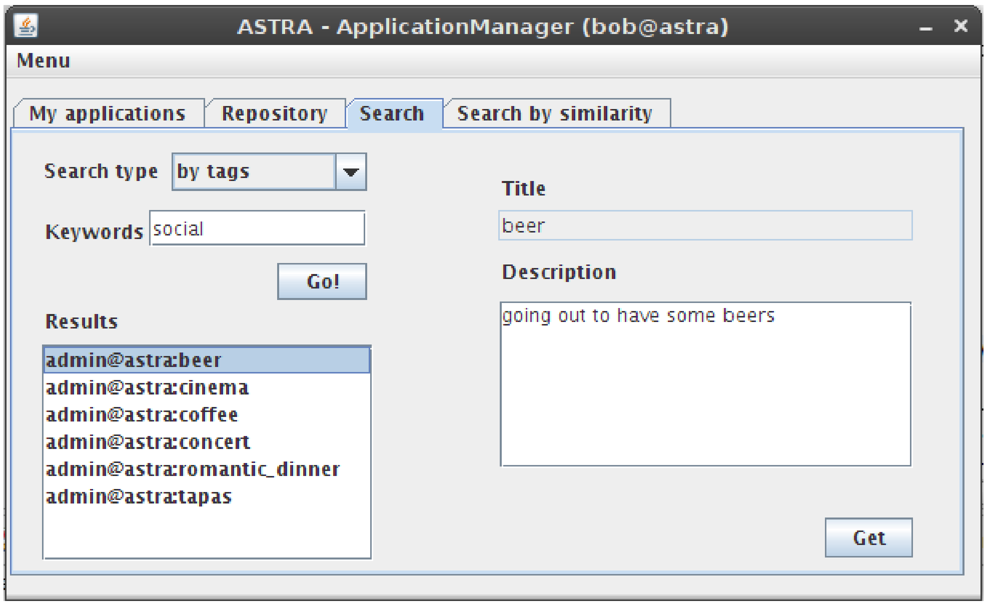
Description

I feel like going for a walk in the park

Tags

social
park
exercise

Figure 4.26: Browsing the repository (screenshot)



Search type: by tags

Keywords: social

Results

admin@astra:beer
admin@astra:cinema
admin@astra:coffee
admin@astra:concert
admin@astra:romantic_dinner
admin@astra:tapas

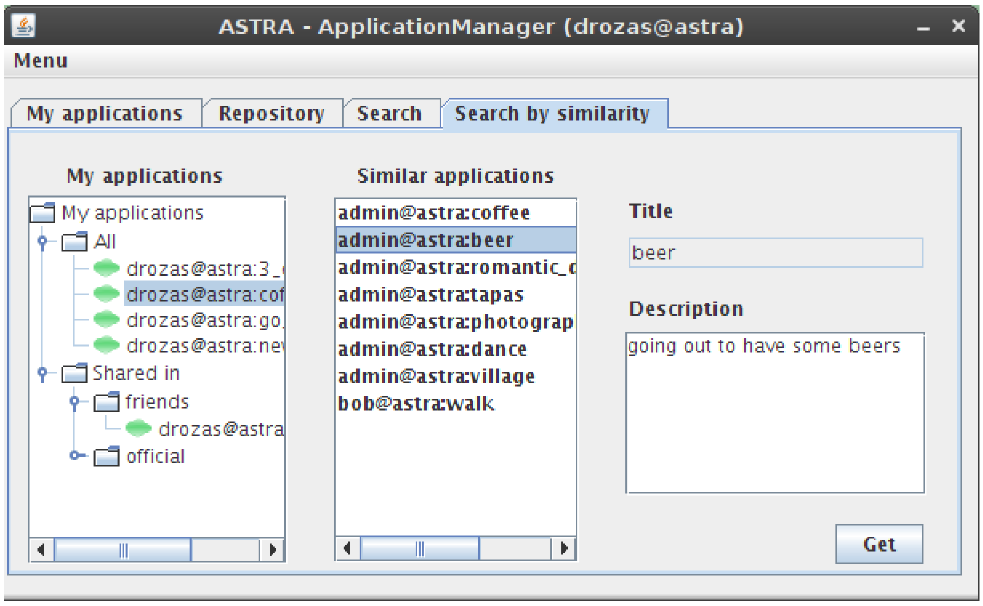
Title

beer

Description

going out to have some beers

Figure 4.27: Searching an application by criteria (screenshot)



My applications

- My applications
  - All
    - drozas@astra:3_
    - drozas@astra:cof
    - drozas@astra:go
    - drozas@astra:ne
  - Shared in
    - friends
      - drozas@astra
    - official

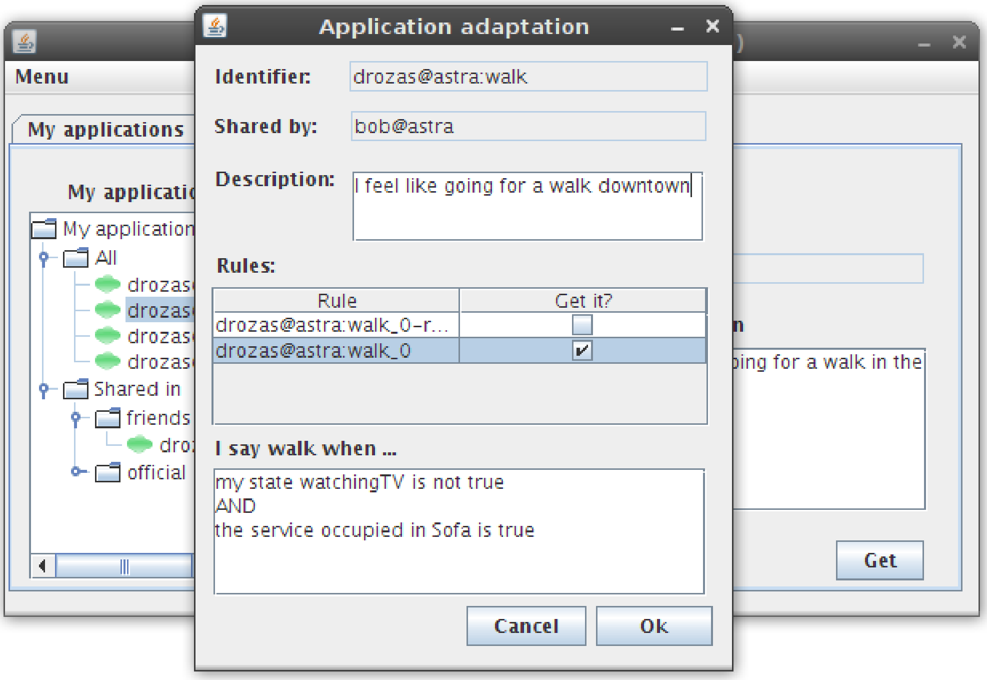
Similar applications

- admin@astra:coffee
- admin@astra:beer
- admin@astra:romantic_c
- admin@astra:tapas
- admin@astra:photograp
- admin@astra:dance
- admin@astra:village
- bob@astra:walk

Title: beer

Description: going out to have some beers

Figure 4.28: Searching an application by similarity (screenshot)



Identifier: drozas@astra:walk

Shared by: bob@astra

Description: I feel like going for a walk downtown

Rules:

| Rule | Get it? |
|---|---|
| drozas@astra:walk_0-r... | ☐ |
| drozas@astra:walk_0 | ✓ |

I say walk when ...

my state watchingTV is not true
AND
the service occupied in Sofa is true

Figure 4.29: Retrieving and adapting an application from the repository (screenshot)

## 4.6 Testing

In this section we will explain shortly the testing processes we carried on for this project. As it was explained in Section 3.1, we have followed a spiral model, therefore we have performed testing tasks for each iteration (see Table 3.1).

### 4.6.1 Functionalities verification

In order to verify the code, we have performed several tests which assure it works accordingly to its expected behavior. This was performed in a low level (after finishing every functionality) and in a high level (after every iteration or before every demonstration). We also stressed on being sure the application responds correctly in extreme cases like:

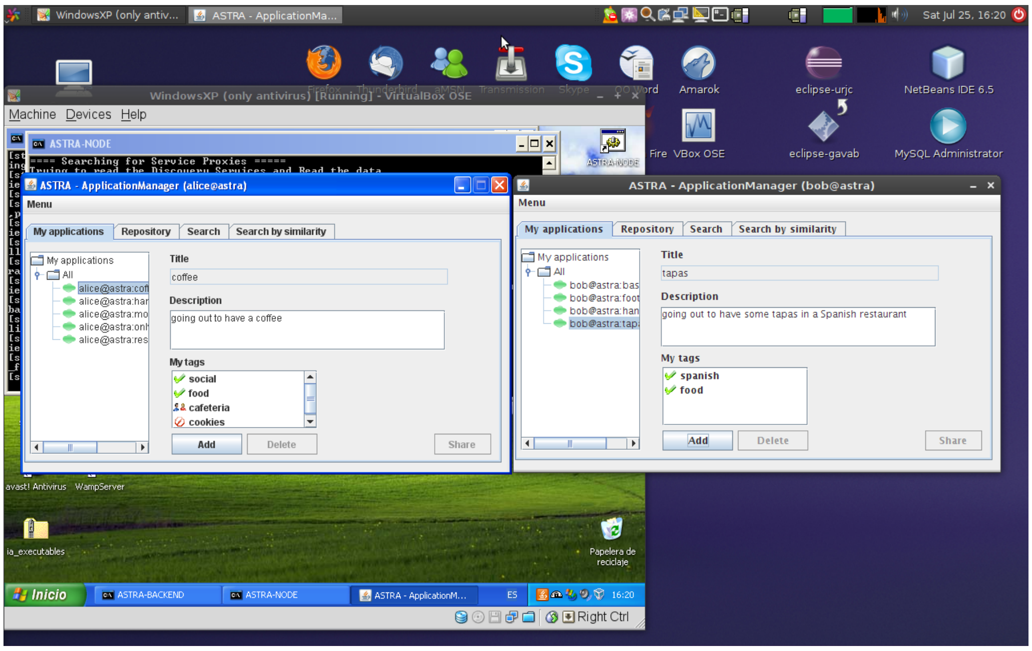
- Lack of local/remote applications.
- No membership in any community.
- The network is down.
- ...

There was also a special interest in assuring all the functionalities related to the visibility of the applications, tags, etc. work properly due to its straight relationship with privacy protection.

As we will see in Section 4.6.4, a preliminary version (iterations 1 and 2) was successfully tested in an users evaluation in December 2008, and another users evaluation is scheduled in October 2009. The current version of the project have passed successfully all the tests we performed and we can assure it offers all the expected functionalities described in the previous sections, so it is ready for being tested in this new users evaluation. The feedback obtained after carrying on that process will be very valuable to improve it in the future.

### 4.6.2 OS compatibility

This project has been totally developed under GNU/Linux, concretely under Ubuntu 8.04 and 8.10. All the bundles created for this project have been tested successfully under different versions of GNU/Linux and Windows XP (see Figure 4.30) for both subcomponents: ASTRA Node and ASTRA Backend.



```
==== Searching for Service Proxies =====
Trying to read the Discovery Services and Read the data
```

Title: coffee

Description: going out to have a coffee

My tags: social, food, cafeteria, cookies

Title: tapas

Description: going out to have some tapas in a Spanish restaurant

My tags: spanish, food

Figure 4.30: ApplicationManager running in GNU/Linux and Windows XP simultaneously

### 4.6.3 Search engine testing

In this section we will explain the process we carried on to evaluate the search by similarity process. Since the origin of the data is totally artificial (it was created by only one person) it does not intend to be a real study, but it explains a possible way to evaluate it once data gathered from a real set of users will be available.

We created a set of 25 applications, with a description and a set of 4 tags for each of them. All these applications were under the ownership of an user called `"admin"` and shared in a community called `"official"`. These applications were divided into 5 groups: “Sport”, “Social”, “Feelings”, “Cultural” and “Location”. We assumed an application can only belong to one of this groups to make the measuring process simpler, but this introduces an error, since the way an application is categorized is subjective and not exclusive. For example, an application `admin@astra:concert` with description “going to a live music concert” was classified into the group “Cultural”, but for other people could be in the group “Social” or even in both. This will be taken into account when evaluating the results, since we will prefer to have a bigger recall even at the expense of loosing precision. The raw data can be found in the Tables B.1, B.2, B.3, B.4 and B.5 in Appendix B.

We created a new user afterwards that owns 5 applications which are similar (in description and tags) to one of each of the groups. This will help us to evaluate if we chose properly the fields to create the query and the fields to perform the query in (see Section 4.4.3). As we explained in the previously referred section, the results are afterwards sorted by score (see Appendix A for more details in scoring in Lucene).

The measures we took were:

- *RD*: Total number of retrieved documents.
- *RDR*: Relevant documents retrieved.
- *ERD*: Total number of existing relevant documents.

With these variables we can calculate the precision and recall percentage:

$$precision = (RDR/RD) * 100$$

$$recall = (RDR/ERD) * 100$$

Before taking the measures and perform the calculations, we defined a small set of testing goals:

- The application with the biggest score should be always the one it was based on. This acts as a "control measure" for the scoring.
- We should have a recall of at least 80%.
- We should have a precision of at least 40%.

The reason why we stressed on the recall values at the expense of the precision is because of the nature of the categorization, as it was explained before.

The measures and calculations for each application can be found in Tables B.6, B.7, B.8, B.9 and B.10 in Appendix B.

Table 4.24 summarizes the calculations by groups and on average.

From this simple evaluation, we can remark:

- It satisfies the preliminary testing goals, with a 96% average in recall and 52.54% average in precision. It also satisfies the "control group" goal in all the cases.

| Group | RD | RDR | ERD | Precision | Recall |
|---|---|---|---|---|---|
| Sport | 9 | 5 | 5 | 55.56% | 100% |
| Social | 10 | 5 | 5 | 50% | 100% |
| Feelings | 10 | 5 | 5 | 50% | 100% |
| Cultural | 7 | 4 | 5 | 57.14% | 80% |
| Location | 10 | 5 | 5 | 50% | 100% |
| Average | 9.2 | 5 | 5 | 52.54% | 96% |

Table 4.24: Summary of the results for search by similarity evaluation

- In most of the cases, the 5 first results belong to the group we were expecting (see Appendix B). This could be useful to establish a threshold in case the precision will decrease too much in the future.
- When there are false positives, in general all of them belong to no more than one or two of the categories (see for instance Table B.8). This can be explained because of the rigid way we have tagged and categorized the applications in contrast with the intrinsic non-exclusive nature of the applications categorization.

It is important to remark that the data was totally artificial, therefore the aim of this small evaluation was just assuring the design decisions point to the right way, and to establish a possible way of measuring in the future.

### 4.6.4 Users evaluation

A preliminary users evaluation after iterations 1 and 2 (see Table 3.1) was performed at NTNU in Trondheim (Norway) on the 13th of December 2008. There were 8 participants, all of them exchange students since those represent one of the main scenarios in ASTRA. A deep analysis is beyond the scope of this document (detailed information can be found in [DC09]), but we will summarize the most important ideas and impressions given by the users, and the way we have used this feedback for developing the current version of the project:

- Participants were positive about the idea of sharing and getting applications.

- The notion of community had a good reception, and they evaluated positively the visibility in terms of community. This feedback has been taken into account in iterations 3 and 4 (i.e.: adding support to browse by communities in local and remote applications).

- The search functionalities needed to be extended. This is one of the main features added in iteration 3 with the integration of a search engine into the repository.

- Users evaluate very positively the possibility of having public and community tags. Users also pointed the necessity of removing them, which is supported in the current version.

- Users evaluate positively the introduction of ontologies. For instance this can be used to support a better categorization. This issue is part of the future work (see Section 5.3), and the bundles developed for this project are already integrated with `OntologyManager`'s interface, so they will work once that functionalities are implemented in that bundle.

- Some privacy issues arise, regarding especially the need of removing rules. For instance, an application may have a rule which provides information about our location that we do not want to share. This has been addressed in the current version, in which we provide mechanisms to visualize and adapt the application (i.e.: discard a rule after visualizing its auto-generated description), so the users can handle its privacy.

- Implementing a search by similarity from a recommended application was pointed as an interesting option. This is addressed in the current version.

A new users evaluation at NTNU including the work performed during iterations 3 and 4 is scheduled in October 2009, once some other ASTRA components will be developed, so they can also be tested in the same evaluation.

## 4.7 Coordination

In this section we will discuss shortly the mechanisms we used in order to coordinate with the rest of teams during the realization of this project. This is specially interesting taking into account the big amount of partners from several countries which are part of the ASTRA project (see Section 1.1).

The employed coordination mechanisms are listed and described briefly below:

- F2F (Face To Face): I have assisted to F2F meetings with members of NTNU and Telenor teams during my stay in Trondheim (Norway) (see Section 5.2 to see time references). This is obviously the richest communication mechanism, and it was specially interesting for the requirements elicitation processes. This has also been the main coordination mechanism with my supervisor at URJC during my stay in Madrid (Spain).

- Teleconferences: I have had weekly teleconference meetings with NTNU, Telenor and CTI members. This has been a very useful mechanism to coordinate the work in team, to discuss the state of the project, to share ideas, etc.

- Wiki: We have used a wiki website (http://www.astra-project.net/wiki) to coordinate the bundles development process. This has been a very useful tool to be aware about other teams bundles and to produce a good documentation.

- Subversion: The project code is hosted in a SVN server at NTNU (http://basar.idi.ntnu.no/svn/astra/). This has been a really useful tool to coordinate the development process, since it allows us to avoid and resolve possible code conflicts, and to have revisions for every code updating.

- E-mail: This supposes the main asynchronous communication mechanism, and it has been very useful to coordinate with all the members of the team. For instance, it has been used to report bugs.

## 4.8 Tools

In this section we will describe briefly the main tools employed for the achievement of this project. There have been a personal interest in the use of free or open-source software due to all the advantages that it offers to us. Therefore, all the tools we have employed (except Jude, which was used under a freeware license) have a license of these characteristics.

The list below, summarizes the most important tools employed for this project:

- Eclipse 3.4.1 as IDE, including the following plugins (all of them free or open source software):
    - Knopflerfish OSGi IDE, version 1.0.16. Plugin to help in OSGi bundles development (see Section 3.2.3.1).
    - Subclipse, version 1.4.8. Provides support for Subversion.
    - Visual Swing, version 0.9. Provides visual GUI design tools.
    - Textlipse, version 1.3.0. Provides $\LaTeX$ support.
- Knopflerfish, version 2.3. Open source OSGi service platform.
- MySQL, version 5.1. Open source RDBMS (see Section 3.2.8).
- PHPMyAdmin, version 3.2. Open source tool to handle MySQL administration through a web interface.
- Apache, version 2.2. Free software HTTP server.
- Dia, version 0.96. Free software program to draw diagrams.
- Jude Community, version 5.4.1. Freeware program for UML modeling.
- VirtualBox, version 3.0.2. Open source virtualizer for x86 hardware.
- GIMP, version 2.6. Free software for image manipulation.

# Chapter 5

# Conclusions

This chapter concludes the report discussing briefly the fulfillment of the objectives and discussing my personal contribution to ASTRA. Some suggestions about future work are also included, as well as a personal evaluation.

## 5.1 Achieved goals

In this report we have detailed the process carried on to develop and integrate a set of bundles to manage awareness applications into ASTRA, including functionalities for sharing, tagging, locating, appropriating and adapting the applications. A new GUI to access this functionalities has also been developed, demonstrating the flexibility of the ASTRA SOA (see Section 3.2.1.2). All the work has been carried on stressing its extensibility, so new functionalities can be easily added in the future as we will see in Section 5.3. Therefore, we can conclude we have successfully fulfilled all the goals stated in Chapter 2.

## 5.2 Contribution

In this section we will discuss briefly my contribution to ASTRA, including also time references.

I started my contribution to ASTRA as part of my summer job for the IDI (Institutt for Datateknikk og Informasjonsvitenskap) at NTNU during the summer of 2008 (July-September). In this period I participated in the analysis, design, implementation and testing of the first version of `RepositoryManager`, `TagManagerBackend` and

`TagManagerNode`. During this term, I was working in close cooperation with another member of the NTNU ASTRA team, and we followed a extreme-programming model (see Section 3.1). The work performed during this period made up iterations 1 and 2 (see Table 3.1).

I resumed my collaboration with ASTRA project in February 2009. During this period I worked in the analysis, design, implementation and testing of `ApplicationManager` and in the extension of the functionalities of other bundles: including search capabilities in `RepositoryManager` and the addition of new functionalities in `TagManagerBackend` and `TagManagerNode`, etc. This made up iterations 3 and 4 (see Table 3.1). Most of the work was developed during my stay in Madrid, and I coordinated using the mechanisms explained in Section 4.7. I also had the opportunity to come back to Norway to work in ASTRA during one month and a half during the summer, which was extremely useful to elicit and implement the last requirements and conclude successfully the project.

## 5.3 Future work

In this section we present a set of possible enhancements for this project:

- The parameters for the searching by similarity process can be adjusted to increase its recall and precision once an evaluation process similar to the one explained in Section 4.6.3 with real user data is carried on.

- The use of ontologies for helping in the application adaptation process can be included once the implementation of the necessary methods in the `OntologyManager` is finished. `ApplicationManager` is already prepared to make use of its services.

- An extension of the GUI could be performed, adding the possibility of publishing applications, join communities, rules edition, etc.

- New browsing methods (by owner, by tags, etc.) could be developed, to allow the user localize applications in new ways.

## 5.4 Personal evaluation

My work for ASTRA has been one of the most enriching experiences of my career. I have had the opportunity to work in a real researching project and to collaborate with teams from several countries. It has also been really enriching in technological terms, since I have increased my knowledge about all the technologies discussed in Chapter 3. It has been really important in personal terms as well, since this experience allowed me to discover my passion for researching. Therefore, I will always be grateful for having had the opportunity to be part of this project.

# Appendix A

# Document scoring in Lucene

In this section we will explain the scoring process employed in our search engine. We use the class `org.apache.lucene.search.DefaultSimilarity` provided by the Lucene library. This explanation is based on [Apa08b] and [Apa08a].

Lucene scoring uses a combination of the Vector Space Model (VSM) of Information Retrieval and the Boolean model to determine how relevant a given Document is to a User's query. In general, the idea behind the VSM is the more times a query term appears in a document relative to the number of times the term appears in all the documents in the collection, the more relevant that document is to the query. It uses the Boolean model to first narrow down the documents that need to be scored based on the use of boolean logic in the Query specification. Lucene also adds some capabilities and refinements onto this model to support boolean and fuzzy searching, but it essentially remains a VSM based system at the heart.

The score for a query $q$ in a document $d$ is computed as follows:

$$score(q,d) = coord(q,d) * queryNorm(q) * \sum_{t\ in\ q} (tf(t\ in\ d) * idf(t)^2 * t.getBoost() * norm(t,d))$$

where

- *tf(t in d)* correlates to the term's frequency, defined as the number of times term *t* appears in the currently scored document *d*. Documents that have more occurrences of a given term receive a higher score. The default computation for *tf(t in d)* in `DefaultSimilarity` is:

$$tf(t\ in\ d) = frequency^{1/2}$$

- *idf(t)* stands for Inverse Document Frequency. This value correlates to the inverse of *docFreq* (the number of documents in which the term $t$ appears). This means rarer terms give higher contribution to the total score. The default computation for *idf(t)* in `DefaultSimilarity` is:

$$idf(t) = 1 + \log(numdocs/(docFreq + 1))$$

- *coord(q,d)* is a score factor based on how many of the query terms are found in the specified document. Typically, a document that contains more of the query's terms will receive a higher score than another document with fewer query terms. This is a search time factor computed at search time.

- *queryNorm(q)* is a normalizing factor used to make scores between queries comparable. This factor does not affect document ranking (since all ranked documents are multiplied by the same factor), but rather just attempts to make scores from different queries (or even different indexes) comparable. This is a search time factor computed at search time. The default computation in `DefaultSimilarity` is:

$$queryNorm(q) = 1/sumOfSquareWeights^{1/2}$$

  The sum of squared weights (of the query terms) is computed by the query `Weight object`. For example, a boolean query computes this value as:

$$sumOfSquareWeights = q.getBoost()^2 * \sum_{t\ in\ q} ((idf(t) * t.getBoost())^2)$$

- *t.getBoost()* is a search time boost of term $t$ in the query $q$ as specified in the query text.

- *norm(t,d)* encapsulates a few (indexing time) boost and length factors:

  – *Document boost*, set by calling `doc.setBoost()` before adding the document to the index.

  – *Field boost*, set by calling `field.setBoost()` before adding the field to a document.

- *lengthNorm(field)*, computed when the document is added to the index in accordance with the number of tokens of this field in the document, so that shorter fields contribute more to the score. *LengthNorm* is computed by the `Similarity` class in effect at indexing.

When a document is added to the index, all the above factors are multiplied. If the document has multiple fields with the same name, all their boosts are multiplied together:

$$norm(t,d) = doc.getBoost()*lengthNorm(field)*\prod_{field\ in\ d\ named\ as\ t}(f.getBoost())$$

# Appendix B

# Search engine - Testing data

In this appendix we detail the data we gathered to perform the evaluation of the search by similarity offered by the search engine. The analysis of this data can be found in Section 4.6.3.

Tables B.1, B.2, B.3, B.4 and B.5 shows the initial set of raw data from which we began our analysis for each group.

| **Sport** | | |
|---|---|---|
| Application identifier | Application description | Tags |
| admin@astra:go_jogging | go jogging to the forest | sport, exercise, run, fit |
| admin@astra:go_to_the_gym | go to the gym to train | fit, health, sport, muscle |
| admin@astra:play_basketball_match | play an amateur basketball match | health, ball, sport, amateur |
| admin@astra:play_football | play a football match | ball, sport, goalkeeper, offside |
| admin@astra:play_handball | play a handball match | sport, fun, ball, amateur |

Table B.1: Search by similarity evaluation - raw data (sport)

| **Social** | | |
|---|---|---|
| Application identifier | Application description | Tags |
| admin@astra:beer | going out to have some beers | beer, fun, social, friends |
| admin@astra:tapas | go to a Spanish restaurant to have some tapas | restaurant, food, spain, social |
| admin@astra:coffee | going to have a coffee | break, social, relax, talk |
| admin@astra:romantic_dinner | going to an Italian restaurant to have a romantic dinner | restaurant, meal, food, social |
| admin@astra:dance | going out to dance in some disco | party, friends, beer, fun |

Table B.2: Search by similarity evaluation - raw data (social)

Tables B.6, B.7, B.8, B.9 and B.10 shows the results returned by the search engine ordered by scored (as they are presented to the user). The application we compare to was based on similar information to the one indicated in **bold** ("control measure", see Section 4.6.3). Applications in *italic* are the ones which belong to the same group

| Feelings | | |
|---|---|---|
| Application identifier | Application description | Tags |
| admin@astra:happy | I am really happy | feeling, nice, love, great |
| admin@astra:hating_you | I think of you, and I am really angry | angry, hate, feelings, furious |
| admin@astra:missing_you | I am missing you so much | love, nostalgy, feeling, lovely |
| admin@astra:sad | I feel sad | lonely, feeling, terrible, nostalgy |
| admin@astra:thinking_of_you | I miss you and I am thinking of you | love, nostalgy, feeling, hug |

Table B.3: Search by similarity evaluation - raw data (feelings)

| Cultural | | |
|---|---|---|
| Application identifier | Application description | Tags |
| admin@astra:go_sightseeing | go sightseeing in our city | tourism, cultural, guide, museum |
| admin@astra:museum | go to a museum | paintings, history, cultural, tourism |
| admin@astra:photography_exposition | going to that new photography exposition | pics, cultural, sightseeing, camera |
| admin@astra:concert | go to a live music concert | music, fun, social, rock |
| admin@astra:cinema | watch a movie in the cinema | movies, social, film, fun |

Table B.4: Search by similarity evaluation - raw data (cultural)

| Location | | |
|---|---|---|
| Application identifier | Application description | Tags |
| admin@astra:girlfriend_home | I am at my girlfriend home | busy, home, place, gf |
| admin@astra:home | I am at home | place, home, location, safe |
| admin@astra:my_parents | I am at my parents home | family, mother, father, place |
| admin@astra:office | I am at work, in my office | work, job, place, office |
| admin@astra:village | I am going to be in my mother village | parents, mother, place, rustic |

Table B.5: Search by similarity evaluation - raw data (location)

we are testing (true positives). An analysis of these results can be found in Table 4.24.

| Sports | | |
|---|---|---|
| Application returned | Field | Score |
| ***admin@astra:play_handball*** | Sport | 2.18 |
| *admin@astra:play_basketball_match* | Sport | 1.4 |
| *admin@astra:play_football* | Sport | 0.64 |
| admin@astra:beer | Social | 0.03 |
| admin@astra:cinema | Cultural | 0.03 |
| admin@astra:concert | Cultural | 0.03 |
| admin@astra:dance | Social | 0.03 |
| *admin@astra:go_jogging* | Sport | 0.03 |
| *admin@astra:go_to_the_gym* | Sport | 0.03 |

Table B.6: Search by similarity evaluation - results for group "Sport"

| **Social** | | |
|---|---|---|
| Application returned | Field | Score |
| ***admin@astra:beer*** | Social | 2.21 |
| *admin@astra:dance* | Social | 0.99 |
| *admin@astra:coffee* | Social | 0.19 |
| *admin@astra:tapas* | Social | 0.18 |
| *admin@astra:romantic_dinner* | Social | 0.15 |
| admin@astra:cinema | Cultural | 0.08 |
| admin@astra:concert | Cultural | 0.08 |
| admin@astra:play_handball | Sport | 0.02 |
| admin@astra:photography_exposition | Cultural | 0.02 |
| admin@astra:village | Location | 0.01 |

Table B.7: Search by similarity evaluation - results for group "Social"

| **Feelings** | | |
|---|---|---|
| Application returned | Field | Score |
| ***admin@astra:thinking_of_you*** | Feelings | 2.4 |
| *admin@astra:missing_you* | Feelings | 0.92 |
| *admin@astra:happy* | Feelings | 0.34 |
| *admin@astra:hating_you* | Feelings | 0.3 |
| *admin@astra:sad* | Feelings | 0.23 |
| admin@astra:home | Location | 0.08 |
| admin@astra:girlfriend_home | Location | 0.07 |
| admin@astra:my_parents | Location | 0.07 |
| admin@astra:office | Location | 0.07 |
| admin@astra:village | Location | 0.06 |

Table B.8: Search by similarity evaluation - results for group "Feelings"

| **Cultural** | | |
|---|---|---|
| Application returned | Field | Score |
| ***admin@astra:museum*** | Cultural | 1.68 |
| *admin@astra:go_sightseeing* | Cultural | 0.91 |
| *admin@astra:photography_exposition* | Cultural | 0.05 |
| *admin@astra:concert* | Cultural | 0.03 |
| admin@astra:go_jogging | Sport | 0.03 |
| admin@astra:go_to_the_gym | Sport | 0.03 |
| admin@astra:tapas | Social | 0.02 |

Table B.9: Search by similarity evaluation - results for group "Cultural"

| **Location** | | |
|---|---|---|
| Application returned | Field | Score |
| ***admin@astra:home*** | Location | 2.88 |
| *admin@astra:girlfriend_home* | Location | 1.28 |
| *admin@astra:my_parents* | Location | 0.73 |
| *admin@astra:office* | Location | 0.2 |
| *admin@astra:village* | Location | 0.19 |
| admin@astra:happy | Feelings | 0.08 |
| admin@astra:hating_you | Feelings | 0.07 |
| admin@astra:thinking_of_you | Feelings | 0.07 |
| admin@astra:missing_you | Feelings | 0.06 |
| admin@astra:sad | Feelings | 0.02 |

Table B.10: Search by similarity evaluation - results for group "Location"

# Appendix C

# Bundles manifests

In this appendix we list the manifests of the bundles developed for this project. The meaning of the most relevant attributes is explained in Section 4.4.1.

## C.1 RepositoryManager's manifest

```
Manifest-Version: 1.0
Export-Package: eu.ist.astra.rm,eu.ist.astra.rm.impl
Bundle-Vendor: IST ASTRA
Bundle-ClassPath: ., lib/lucene-2_4_1_svn.jar
Bundle-Version: 2.0.0
Bundle-Name: RepositoryManager
Bundle-Activator: eu.ist.astra.rm.impl.Activator
Bundle-Description: Repository Manager bundle which manages the sharing
process in the back-end
Import-Package:
org.osgi.framework,
eu.ist.astra.persistency,
eu.ist.astra.cm, eu.ist.astra.tmbe,
eu.ist.astra.em, eu.ist.astra.em.events,
javax.xml.parsers;resolution:=optional,
javax.xml.transform;resolution:=optional,
javax.xml.transform.dom;resolution:=optional,
javax.xml.transform.stream;resolution:=optional,
org.w3c.dom;resolution:=optional
Bundle-SymbolicName: RepositoryManager
```

## C.2 TagManagerBackEnd's manifest

```
Manifest-Version: 1.0
Export-Package: eu.ist.astra.tmbe
Bundle-Vendor: Astra
Bundle-ClassPath: .
Bundle-Version: 2.0.0
Bundle-Name: TagManagerBackEnd
Bundle-Activator: eu.ist.astra.tmbe.impl.Activator
Bundle-SymbolicName: TagManagerBackEnd
Import-Package:
org.osgi.framework,
eu.ist.astra.persistency,
eu.ist.astra.cm, eu.ist.astra.em,
eu.ist.astra.em.events
Bundle-ManifestVersion: 2
```

## C.3 TagManagerNode's manifest

```
Manifest-Version: 1.0
Export-Package: eu.ist.astra.tmn
Bundle-Vendor: Astra
Bundle-ClassPath: .
Bundle-Version: 2.0.0
```

```
Bundle-Name: TagManagerNode
Bundle-Activator: eu.ist.astra.tmn.impl.Activator
Bundle-SymbolicName: TagManagerNode
Import-Package:
org.osgi.framework,
eu.ist.astra.persistency
```

## C.4 ApplicationManager's manifest

```
Manifest-Version: 1.0
Bundle-Vendor: ASTRA
Bundle-ClassPath: ., lib/grouplayout.jar
Bundle-Version: 2.0.0
Bundle-Name: ApplicationManager
Bundle-Activator: eu.ist.astra.am.Activator
Bundle-SymbolicName: ApplicationManager
Import-Package:
org.osgi.framework,
eu.ist.astra.aam, eu.ist.astra.rfm,
eu.ist.astra.rfm.proxies, eu.ist.astra.ontologymanager.api,
eu.ist.astra.tmn,
eu.ist.astra.awarenessmanager,
javax.swing;resolution:=optional,
javax.swing.event;resolution:=optional,
javax.swing.table;resolution:=optional,
javax.swing.text;resolution:=optional,
javax.swing.text.html;resolution:=optional,
javax.swing.tree;resolution:=optional
```

# Appendix D

# OSGi bundles configuration

In this appendix we present the configuration files (`xinit.args`) necessary to run the bundles in the proper order for the Backend and the Nodes. These configuration files are read by the framework on the startup process, so it can decides which bundles to start and in which order.

## D.1 OSGi - Backend configuration

```
#
# Generated from template.xargs Knopflerfish release 2.0.1
# drozas: the latest information about the configuration can
# be found at: http://www.astra-project.net/wiki/ConfigurationForBackEnd
#
# load common properties
-xargs props.xargs

# Prefix for searching for bundle URLs from console or command line
-Dorg.knopflerfish.gosg.jars=file:ASTRA/
#-Deu.ist.astra.cm.proxy.endpoint=
http://localhost:8080/axis/services/CommunityManager

-init

#extra Initializations for ASTRA
```

```
-istart jsdk/jsdk-2.2.jar
-istart log/log_all-2.0.1.jar
-istart cm/cm_all-2.0.0.jar
-istart commons-logging/commons-logging_all-2.0.0.jar
-istart http/http_all-2.0.0.jar
-istart axis-osgi/axis-osgi_all-0.1.0.jar

-istart RemoteFrameworkManager-2.0.0.jar
-istart EventsManager-2.0.0.jar
-istart PersistencyManager-2.0.0.jar
-istart CommunityManagerAPI-2.0.0.jar
-istart UserManager-2.0.0.jar
-istart CommunityManager-2.0.0.jar
-istart AwarenessApplicationManagerBackend-2.0.0.jar
-istart TagManagerBackEnd-2.0.0.jar
-istart RepositoryManager-2.0.0.jar
-launch
```

## D.2 OSGi - Node configuration

```
#
# Generated from template.xargs
# Knopflerfish release 2.0.1
#
# drozas: the latest information about the configuration can
#be found at: http://www.astra-project.net/wiki/ConfigurationForNode
# load common properties
-xargs props.xargs

# Prefix for searching for bundle URLs from console or command line
-Dorg.knopflerfish.gosg.jars=file:ASTRA/
#You may want to change this parameter to refer to another server
-Deu.ist.astra.cm.proxy.endpoint=
http://localhost:8080/axis/services/CommunityManager
```

```
-Deu.ist.astra.BackEndAddress=http://localhost:8080/
-Deu.ist.astra.default.user=john@astra

-init

#extra Initializations for ASTRA
-istart jsdk/jsdk-2.2.jar
-istart cm/cm_all-2.0.0.jar
-istart commons-logging/commons-logging_all-2.0.0.jar
-istart http/http_all-2.0.0.jar
-istart axis-osgi/axis-osgi_all-0.1.0.jar
-istart jena_bundle.jar

-istart RemoteFrameworkManager-2.0.0.jar
-istart EventsManager-2.0.0.jar
-istart OntologyManager-2.0.0.jar
-istart PersistencyManager-2.0.0.jar
-istart CommunityManagerAPI-2.0.0.jar
-istart CommunityManagerProxy-2.0.0.jar
-istart UserManagerProxy-2.0.0.jar
-istart UserManagerAPI-2.0.0.jar
-istart AwarenessManager-2.0.0.jar
-istart AwarenessApplicationManager-2.0.0.jar
-istart ServiceProxyManager-2.0.0.jar
-istart UPnPServiceProxy-2.0.0.jar
-istart ContextManager-2.0.0.jar
-istart TagManagerNode-2.0.0.jar
#-istart ASTRATester-2.0.0.jar
-istart ApplicationManager-2.0.0.jar

-launch
```

# Appendix E

# GNU Free Documentation License

## Preamble

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 4. MODIFICATIONS

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do

## 5. COMBINING DOCUMENTS

any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

## 10. FUTURE REVISIONS OF THIS LICENSE

## 11. RELICENSING

# Bibliography

[Apa08a] Apache Lucene Team. Formula for scoring in lucene. *http://lucene.apache.org/java/240/api/org/apache/lucene/search/Similarity.html*, 2008.

[Apa08b] Apache Lucene Team. Understanding scoring in lucene. *http://lucene.apache.org/java/240/scoring.html*, 2008.

[Bar09] Bartlett, Neil. *OSGi in practice.* CC-E-books, 2009.

[Bo01] Beedle, Mike Beck, Kevin and others. The agile manifesto. *http://agilemanifesto.org/*, 2001.

[BP09] Calemis, Ioannis Berg, Erik and Pérez, Alfredo. Astra soa white paper. *http://www.astra-project.net*, 2009.

[CM07] Calemis, Ioannis and Mavrommati, Irene. Preliminary requirements and approach for tools that configure pervasive awareness applications: the astra case. *http://www.astra-project.net*, 2007.

[Con06] Conradi, Reinard. Astra: Awareness services and systems - towards theory and realization. *http://www.idi.ntnu.no/grupper/su/astra.html*, 2006.

[DC09] Pérez, Alfredo Divitini, Monica and Cassens, Jörg. Design and evaluation of a repository for sharing pervasive awareness applications. *http://www.astra-project.net*, 2009.

[GM05] Gallego, Micael and Montalvo, Soto. *Interfaces gráficas en Java.* Centro de estudios Ramón Areces, 2005.

[Hai04] Haiges, Sven. Osgi tutorial. *http://www.knopflerfish.org/tutorials/*, 2004.

[Le 02] Le Hégaret, Philippe. The w3c document object model (dom). *W3C*, 2002.

[Lyb07] Lybarger, Rob. Mvc in a java/swing application. *http://www.developer.com/design/article.php/3678856*, 2007.

[McC08] McCabe, Francis G. Reference architecture for service oriented architecture. *http://docs.oasis-open.org/*, 2008.

[PS07] Rogers, Yvonne Preece, Jenhy and Sharp, Helen. *Interaction design: beyond human-computer interaction (2nd edition)*. Wiley, 2007.

[San05] Sanjay, Addicam V. Overview of agile management and development methods. *The PROJECT PERFECT White Paper Collection*, 2005.

[Sun08] Sun Microsystems. Java tutorials: Concurrency in swing. *http://java.sun.com/docs/books/tutorial/uiswing/concurrency/*, 2008.