Development and integration of an awareness applications manager into ASTRA

Presentación de Proyecto de Fin de Carrera

David Rozas Domingo

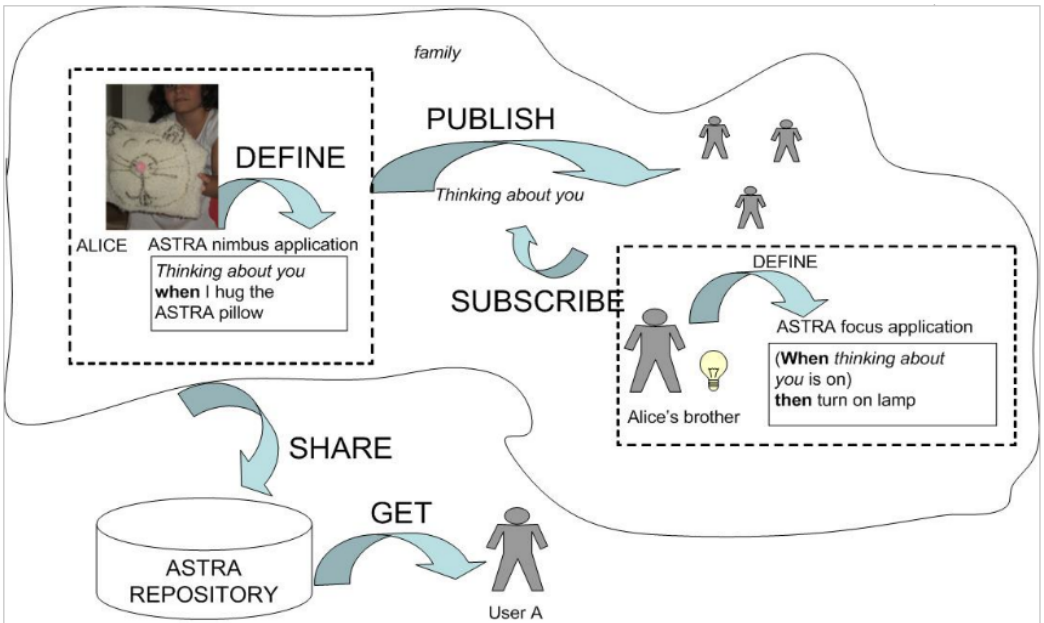
Tutoras:
Soto Montalvo (Universidad Rey Juan Carlos)
Monica Divitini (Norges Teknisk-Naturvitenskapelige Universitet)

What is ASTRA? - The big picture

- ASTRA: Awareness Services and Systems - Towards Theory and Realization.
- Researching awareness systems and services that are used for social purposes.
- Computer-mediated communication systems that help individuals or groups build and maintain a peripheral awareness of each other.
- They offer low effort and non-intrusive communication.
- Organizations involved: CTI, Telenor, Philips, NTNU, etc.

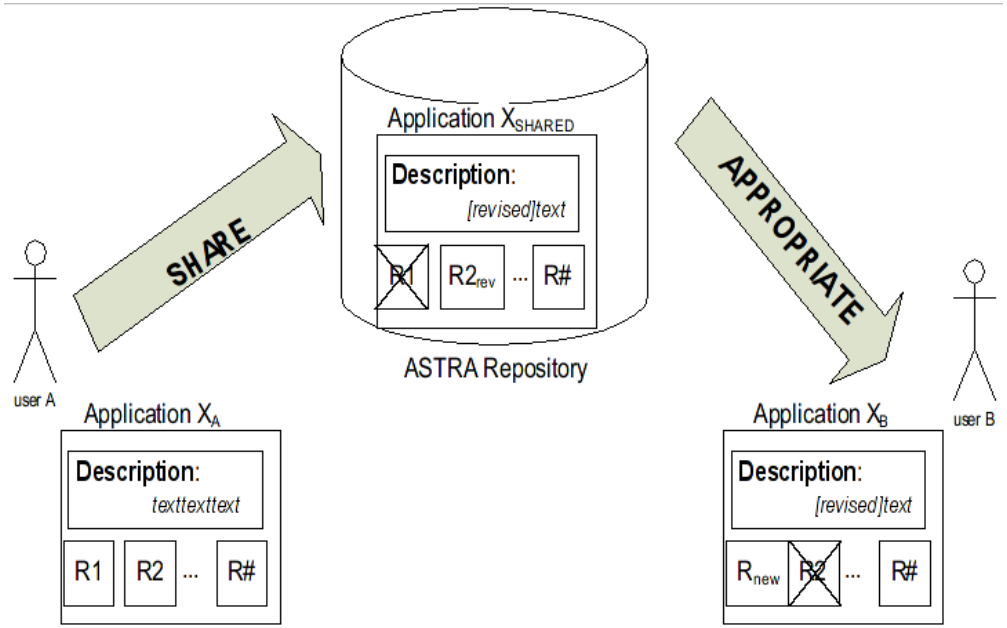
Some key concepts

- ASTRA Application: transform the services provided by the system into awareness applications. Two types:
  - *Nimbus*: to make available awareness information to other users.
  - *Focus*: to decide what information we are interested in and in which way we want to receive it.



ASTRA applications example: video

- Idea: 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.



Objectives

- Create a main repository for applications, where the users can browse, share and retrieve them. Taking into account:
  - Flexible sharing process: choosing rules and communities.
  - Application adaptation during the retrieval process.
  - Changes needed as transparent as possible to the user
  - Add a mechanism to allow searching applications: by different criteria (tags, description, type, etc.) and by recommendation based on the similarity with respect to a local application.
- Create a system which allows the users to tag the applications. Taking into account different scopes:
  - Private tags.
  - Community tags.
  - Public tags.

Objectives

- Create a GUI which allows the user to carry out these operations. 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.

Methodology

- Agile software development methodology: iterative development, frequent inspection and adaptation, self-organization, etc.
- Following a spiral model.
- Fits properly with the researching nature of the project and the fact that new requirements arise continuously.

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: Iterations during the project

Involved technologies

- SOA: Service Oriented Architecture.
- Paradigm for organizing and utilizing distributed capabilities
- Resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation
- SOAP is a lightweight protocol for exchanging structured information in a decentralized, distributed environment. The implementation we have used is Apache Axis (open source).

ASTRA SOA

- The backbone of ASTRA: the platform that offers all the services (awareness, ontologies, the ones developed for this project, . . .)
- They are grouped into two subsystems: ASTRA Node and ASTRA Backend (Client-Server model).
- Therefore it is important to distinguish between the local and remote nature connection when consuming other bundles services: limitations in the type of objects, problems with the network, etc.

ASTRA SOA bundles' services used in this project:

- `UserManager` (Backend): Manage users, their profiles and their identities.
- `CommunityManager` (Backend): Connect virtual community representations in within which users can share awareness information.
- `AwarenessManager` (Nodes): Connection between low level user-system interaction and the high level concepts related to them (i.e.: rules engine kernel).
- `AwarenessApplicationManager` (Nodes): Store and manage local awareness applications.

- `OntologyManager` (Nodes): Manage, look-up and extend ontologies. It is executed in the Nodes.
- `PersistencyManager` (Both): Storage functionalities.
- `RemoteFrameworkManager` (Both): Facilities to consume remote bundles services.
- `EventsManager` (Both): Communicate events between bundles.

OSGi

- OSGi (Open Services Gateway initiative) is a flexible framework, which provides a standardized environment for service deployment and operation.
- The components (bundles) can be remotely installed, started, stopped, updated and uninstalled without requiring a reboot.
- It is a collaborative environment: bundles run in the same VM and can actually share code.
- It enforces a clean service oriented design approach for ASTRA, with a clear distinction between interfaces and implementation.
- We used OSGi-Knoplerfish implementation (BSD style license).

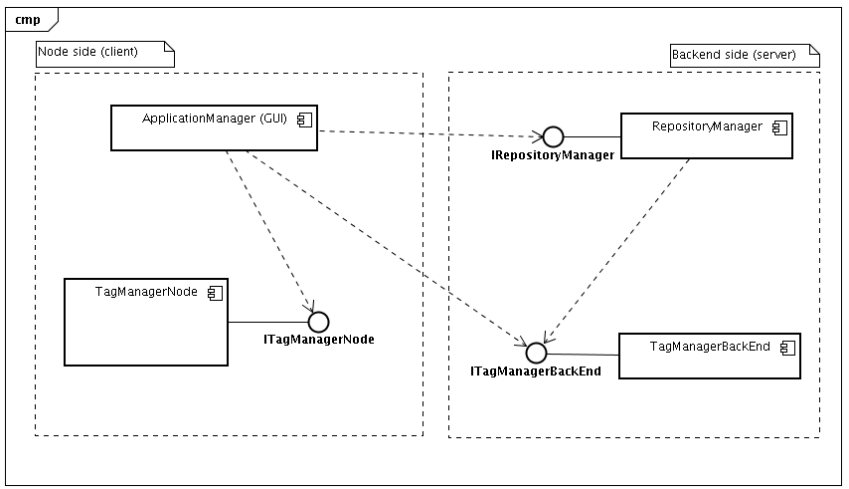
Lucene

- Apache Lucene is an open source information retrieval library.
- It is just an indexing and search library, not an application.
- It allowed us to create a search engine integrated in one of the bundles to offer searching capabilities in it.
- It is open source, it has a great performance, it is cross-platform, easily to extend, etc.

Design

The functionality was divided into the following components (bundles):

- `RepositoryManager`: share, retrieve, store and search applications.
- `TagManagerBackEnd`: manage public and community tags.
- `TagManagerNode`: manage private tags.
- `ApplicationManager`: interaction with the user and connection with the proper bundles to satisfy his requests.

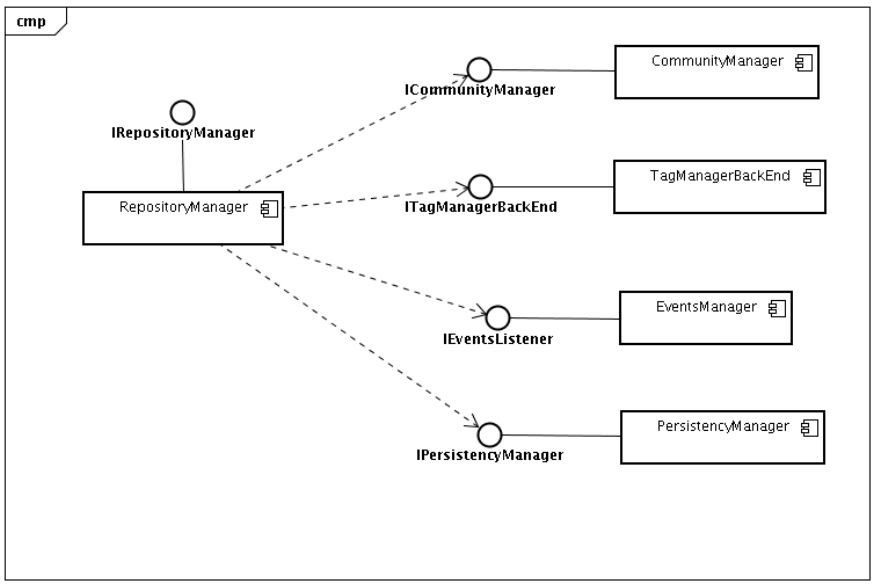


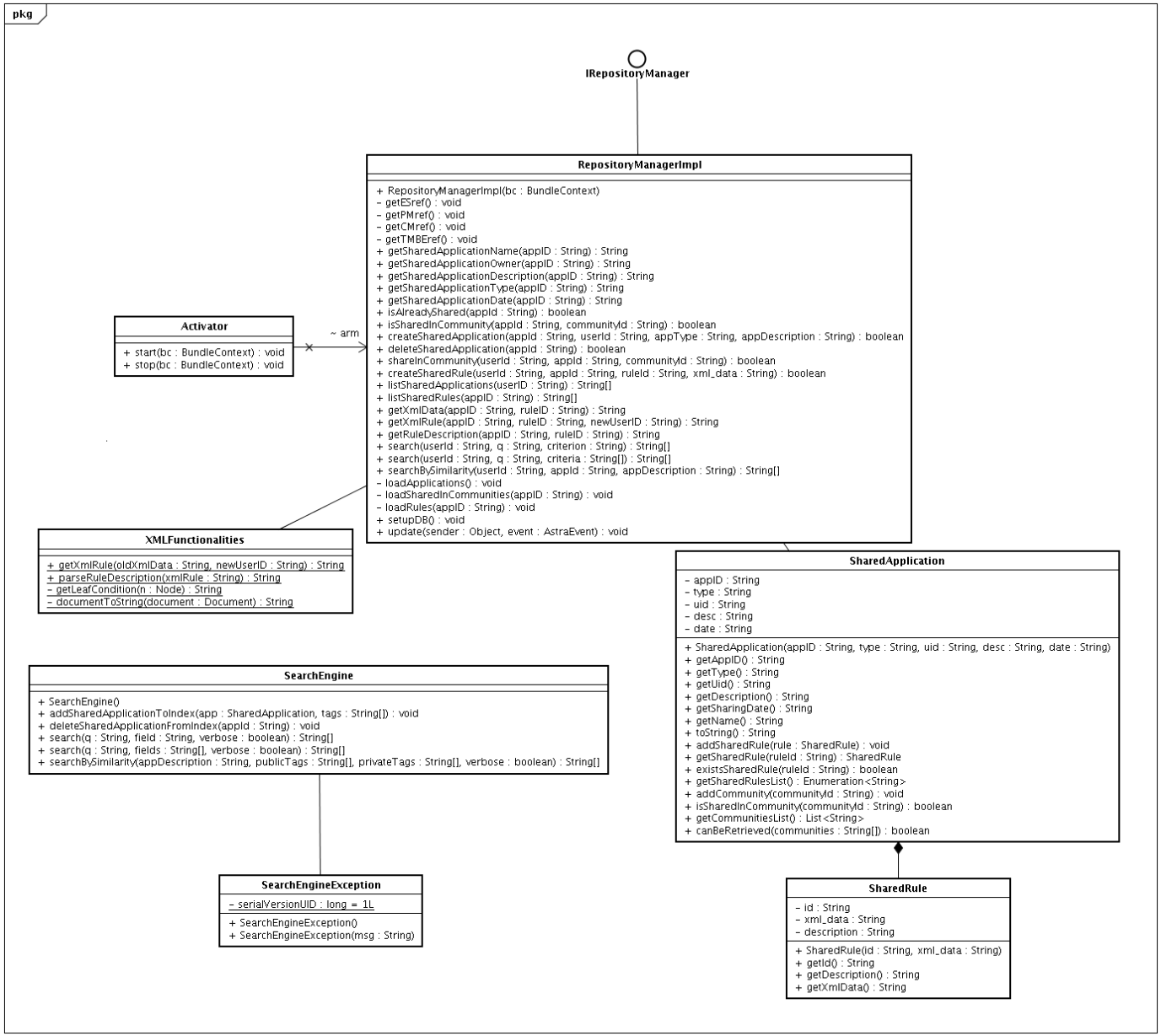
RepositoryManager

- One of the key pieces:
  - Offer services to share applications.
  - Offer services to retrieve applications.
  - Offer services to search applications by criteria (description, tags, type or any)
  - Offer services to search applications by similarity (with respect to a local application).
  - Storage of shared applications.
  - Functionalities to adapt the application.
  - etc.
- It is executed in the Backend.

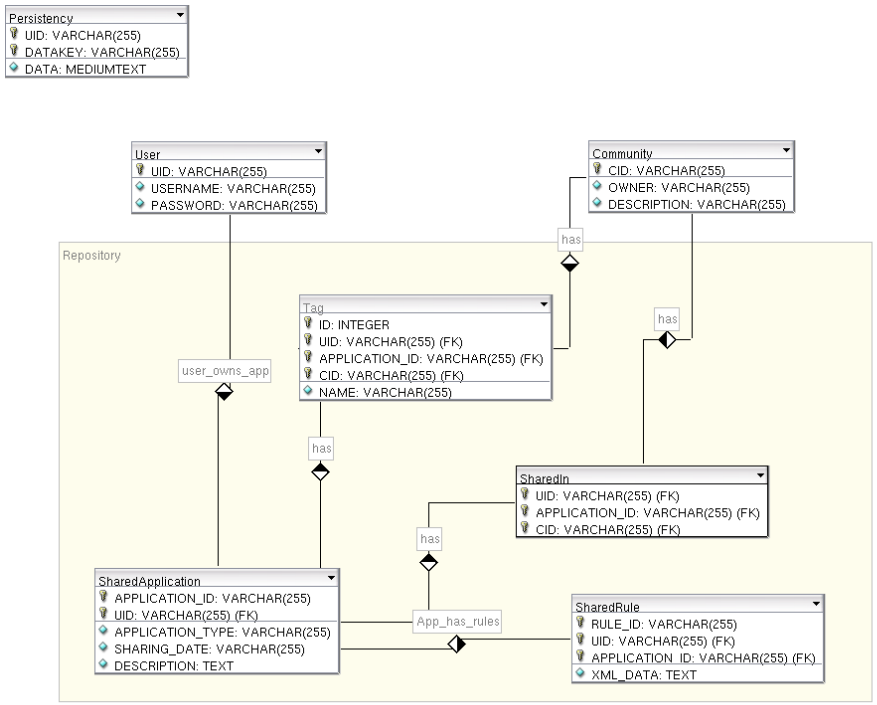
Connection with other bundles:

- `CommunityManager`: retrieve information about relationship between users, communities & applications. Ex.: assure visibility.
- `TagManagerBackEnd`: analyze tags to create search engine index.
- `EventsManager`: keep track of events in TMBE. Ex.: search engine index updated dynamically.
- `PersistencyManager`: store data in the DB.




pkg
IRepositoryManager
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
~ arm
XMLFunctionalities
+ getXmlRule(oldXmlData : String, newUserID : String) : String
+ parseRuleDescription(xmlRule : String) : String
- getLeafCondition(n : Node) : String
- documentToString(document : Document) : 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
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)
SharedRule
- id : String
- xml_data : String
- description : String
+ SharedRule(id : String, xml_data : String)
+ getId() : String
+ getDescription() : String
+ getXmlData() : String

Storage in DB:



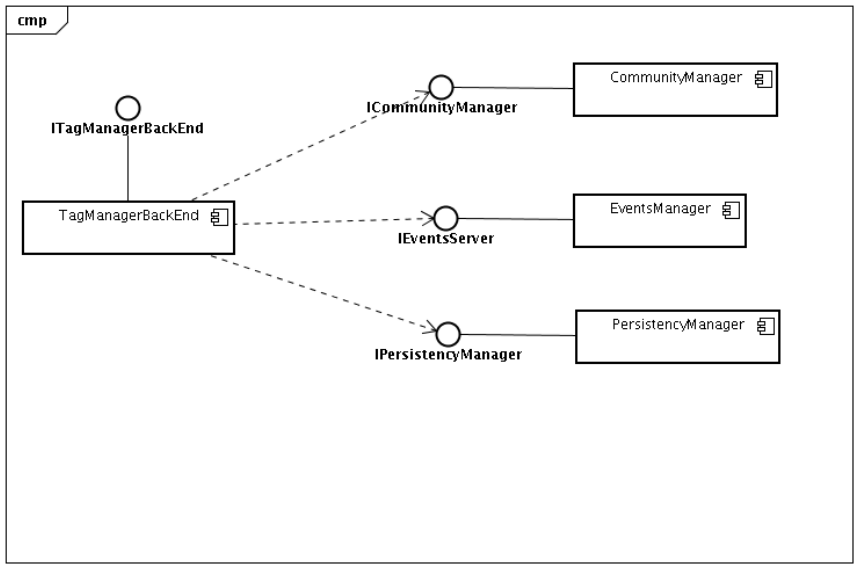
+ a synchronized copy in local memory to increase retrieving operations performance (drawback: duplicated create/delete operations).

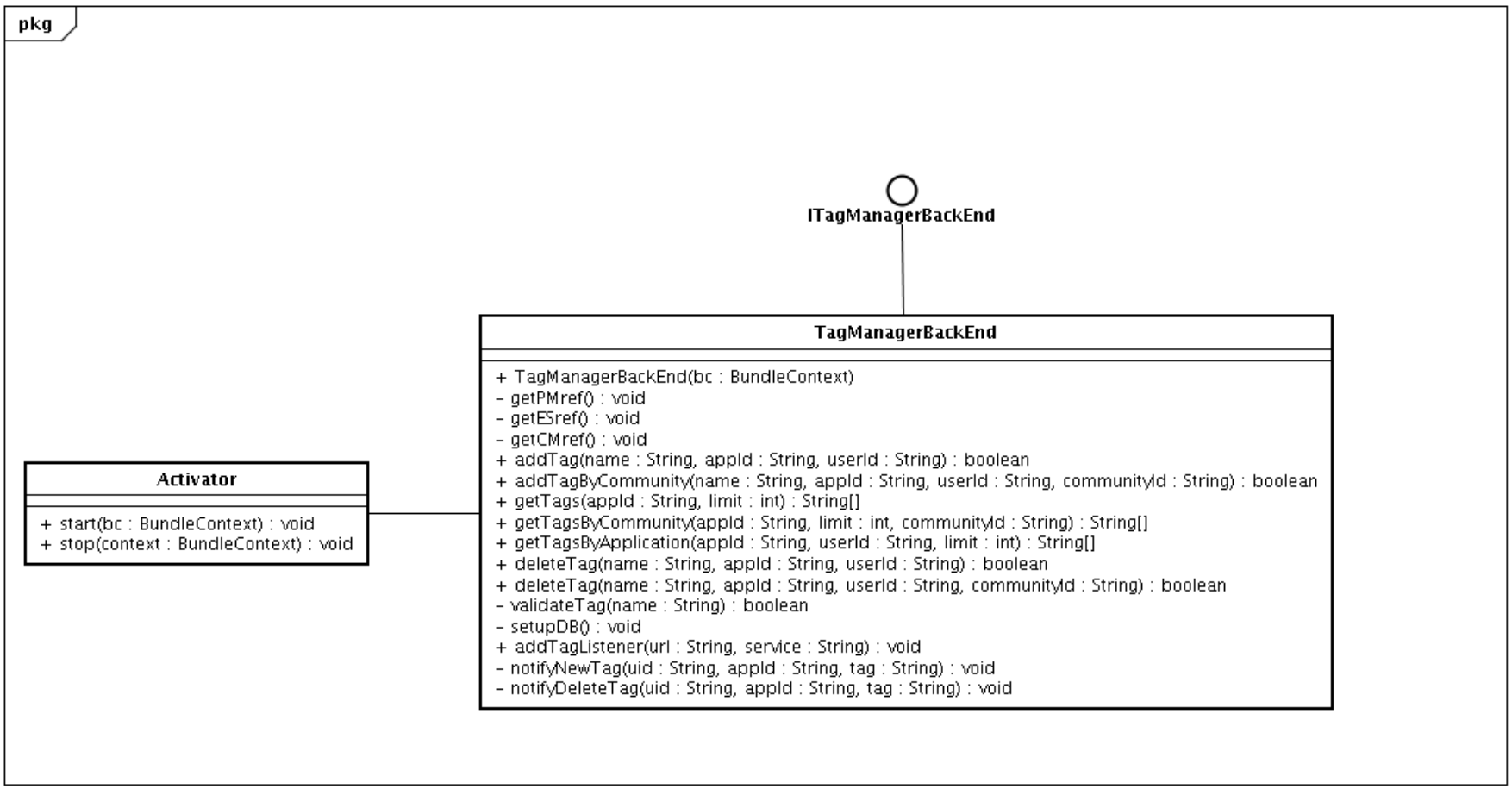
TagManagerBackEnd

- Offer services to add and delete public and community tags.
- Offer services to retrieve those tags in different and flexible ways: by visibility, by communities, only public ones, etc.
- It is executed in the Backend.

Connection with other bundles:

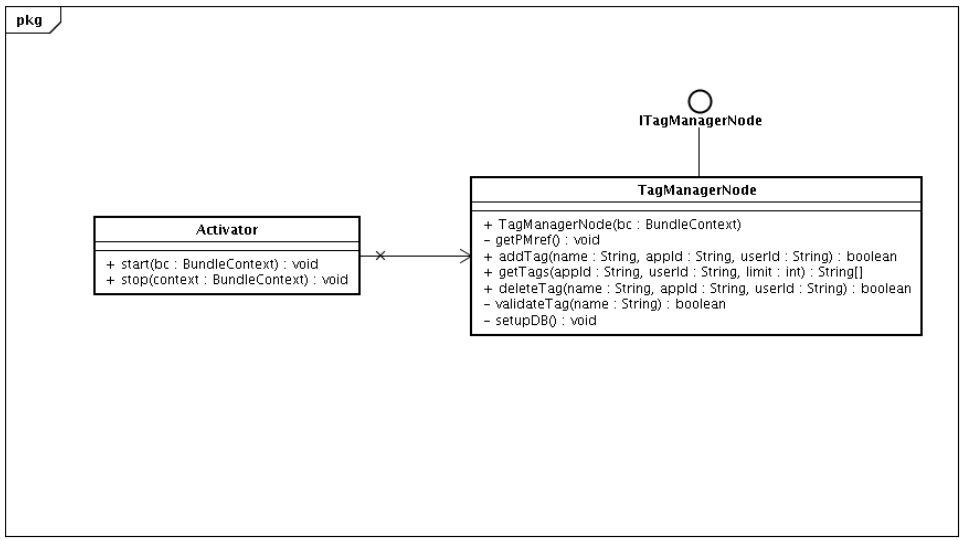
- `CommunityManager`: retrieve information about relationship between users, communities & tags. Ex.: assure visibility.
- `EventsManager`: give feedback of the events.
- `PersistencyManager`: store data in the DB.




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

TagManagerNode

- Offer services to add and delete private tags.
- Offer services to retrieve those tags in different and flexible ways.
- It is executed in the Nodes.



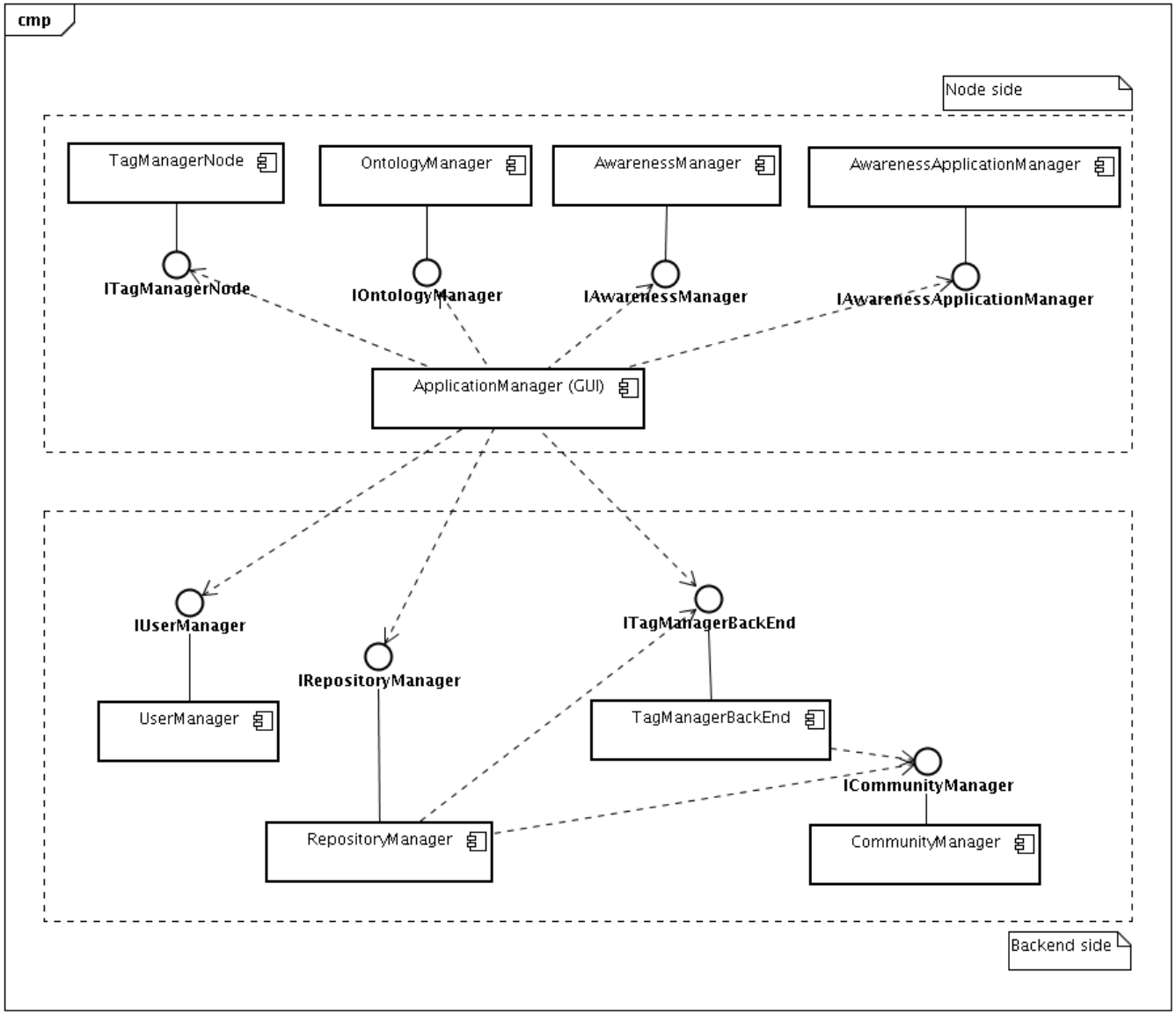
ApplicationManager

- It is in charge of the interaction with the user by offering an intuitive GUI.
- Responsible of the connection with other ASTRA bundles services based on that interaction.
- It is executed in the Nodes.
- It is connected with bundles from both edges: Node and Backend.

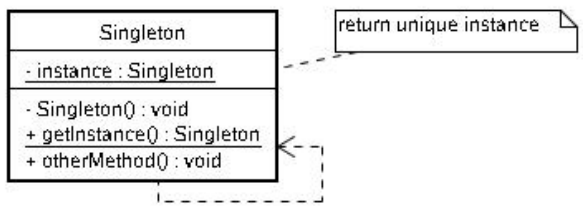
Connection with other bundles:

- Local bundles:
  - `AwarenessApplicationManager`: information about local applications.
  - `AwarenessManager`: information about local rules.
  - `TagManagerNode`: to manage the private tags.
  - `OntologyManager`: to assist the user in the application adaptation process using ontologies[1].
- Remote bundles:
  - `UserManager`: users authentication.
  - `CommunityManager`: information about the communities joined by the user.
  - `TagManagerBackend`: to manage the public and community tags.
  - `RepositoryManager`: to share and retrieve applications with the rest of the users.

[1] Future work.


cmp
Node side
TagManagerNode
OntologyManager
AwarenessManager
AwarenessApplicationManager
ITagManagerNode
IOntologyManager
IAwarenessManager
IAwarenessApplicationManager
ApplicationManager (GUI)
IUserManager
IRepositoryManager
ITagManagerBackEnd
UserManager
TagManagerBackEnd
ICommunityManager
RepositoryManager
CommunityManager
Backend side

- It 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.
- Some of the classes make use of a singleton pattern, that make the class itself responsible for keeping track of its sole instance, ensuring that no other instance can be created.



- Controller:
  - It is represented by the class `ApplicationManagerController` (singleton).
  - Keep track of references to user interface components.
  - Provide a set of methods that other components can directly call in their event handler.
- Model:
  - It is represented by the class `ApplicationManagerModel` (singleton).
  - Manage the references to the rest of the bundles.
  - Work as an "stub container" to make use of the services provided by those.
  - Take care of the session data, i.e.: the user identifier.
- View:
  - It is represented by several classes whose task consist of presenting the information to the user.
  - Includes all the classes which represent the windows (i.e.: `MainWindow`, `LoginWindow`, etc.) and all the classes that extend some of the graphical components (i.e.: `TreeRenderer`).

Implementation

Many interesting implementation details. . . but we will focus on two:

- Search engine implementation (Lucene).
- Application adaptation & rules description creation (XML/DOM).

Search engine

First of all, we need to implement the index:

- From Lucene concepts...
  - `Document`
  - `Fields`
- ...to ASTRA concepts:
  - Astra Application
  - some Astra Application attributes (description,type, tags, etc.)
- Type of index: `RAMDirectory`, more efficient and no need of persistence.

And we have to decide the way we perform the queries:

- Querying by criteria is almost straight:
  - Any = keywords in (description, type, tags, ...)
  - Description = keywords in (description)
  - ...
- But in the "search by similarity" case, testing was needed. Finally:
  - {Public + visible community tags} in (description, tags)

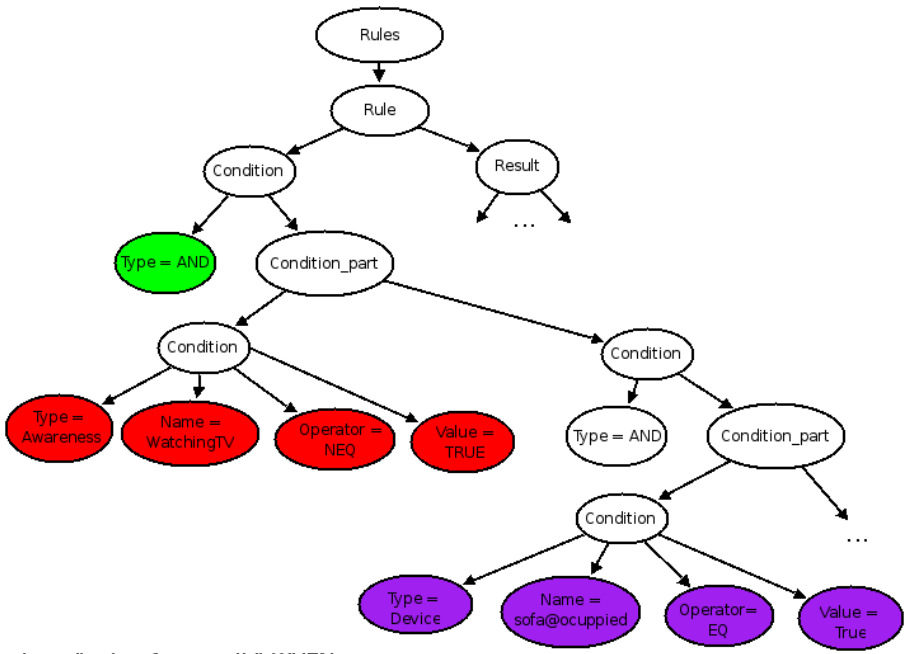
Application adaptation & rules description creation

- Some are performed by the user: choosing rules, changing description, etc.
- Some are performed in a transparent way:
  - It required more analysis.
  - I.e.: Modify the ownership of the rules internally for certain nodes.
  - Implemented using DOM.

```
<?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>
```

- A similar approach was taken to create a description of the rule based on the XML file that represents it.
- I.e.: creating a human readable description for a rule while analyzing recursively the tree.



Testing

Testing in different levels:

- Functionalities verification (low and high level): visibility, network is down, extreme cases, etc.
- Different OS compatibility: GNU/Linux, Windows XP, etc.
- Preliminary users evaluation (iterations 1 & 2):
  - Performed at NTNU (Trondheim) in December 2008.
  - Users were very positive about the idea of sharing/getting applications, about the notion of community, etc.
  - Very useful for adding new functionalities in iterations 3 & 4: searching by criteria, searching by similarity, application adaptation, etc.
- New user evaluation (including iterations 3 & 4) is scheduled in October 2009.

Ex.: search engine testing (similarity)

- Set of 25 applications, with a description and a set of 4 tags for each of them.
- 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.
- We calculate precision and recall.

$$precision = (Relevant\ docs / Total\ retrieved\ docs) * 100$$

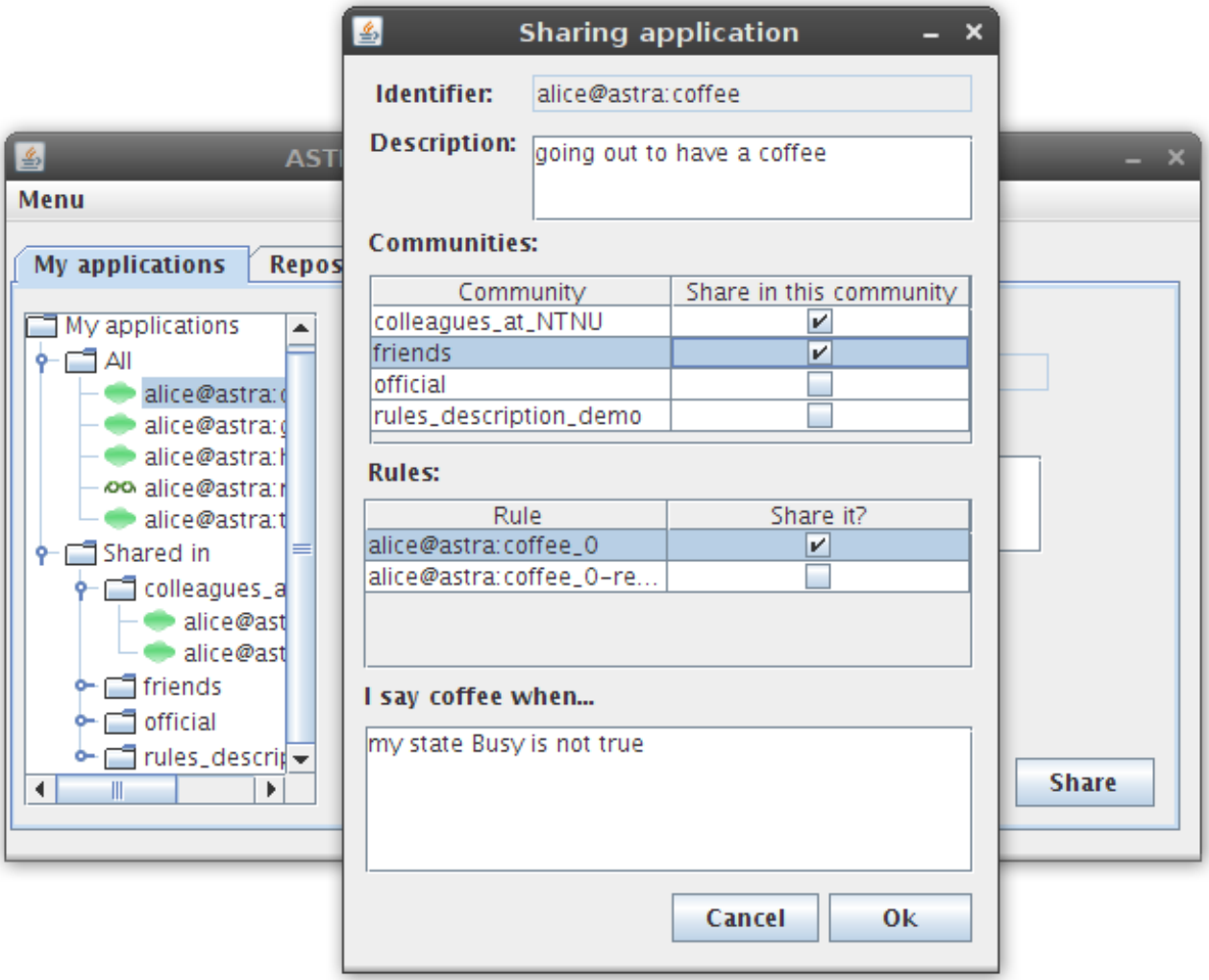
$$recall = (Relevant\ docs / Existing\ relevant\ docs) * 100$$

- Expected values: $p = 40\%$, $r = 80\%$.
- Results were quite positive: $p = 52\%$, $r = 96\%$ . . . but this is based on artificial data.
- Useful to check design decisions point to the right way and to set guidelines for October's evaluation.

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: Summary of the results for search by similarity evaluation

Demonstration



Coordination

The main mechanisms to coordinate with the rest of the teams were:

- F2F: F2F meetings with members of NTNU and Telenor teams during my stay in Trondheim (Norway) and with my supervisor in Madrid. It was the richest communication mechanism, and it was specially interesting for the requirements elicitation processes.
- Teleconferences: Weekly teleconference meetings with NTNU, Telenor and CTI members. Very useful mechanism to coordinate the work in team, to discuss the state of the project, to share ideas, etc.
- Wiki: Wiki website (http://www.astra-project.net/wiki) to coordinate the bundles development process and to create documentation.

- Subversion: The project code is hosted in a SVN server at NTNU (http://basar.idi.ntnu.no/svn/astra/). Very useful 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: Useful to coordinate with all the members of the team in an asynchronous way. For instance, it has been used to report bugs.

Conclusions & future work

- Achieved goals:
  - Successful development and integration of a set of bundles to manage awareness applications into ASTRA, including functionalities for sharing, tagging, locating, appropriating and adapting the applications.
  - New GUI, which demonstrates the flexibility of ASTRA SOA.
  - Stressing its extensibility, so new functionalities can be easily added.
- Future work:
  - Adjust searching by similarity parameters, once we have real users data.
  - Use of ontologies for helping in the application adaptation process, once `OntologyManager` services are implemented (`ApplicationManager` is already prepared).
  - Extend the GUI: new browsing capabilities, join communities, rules edition, etc.

- Contribution & Personal evaluation:
  - I started my contribution to ASTRA as part of my summer job for NTNU during the summer of 2008: iterations 1 & 2.
  - I resumed my collaboration with ASTRA project in February 2009, working from Madrid: iterations 3 & 4.
  - I came back to Norway to work in ASTRA during one month and a half during the summer: very useful to elicit and implement last requirements.
  - It has been a very enriching experience: opportunity of working in a real researching project collaborating with teams from several countries, in technological terms, etc.

Thanks!

¡Gracias!

Takk!