6
Description of the Tool Support

In this section, we sustain our approach with a tool developed, as a web application, which supports the identification, classification, sharing, search, recommendation, retrieval and subscription of agent components.

When the application begins, the agent platform starts with all agents that will be running in the application, like is shown in the screenshot of Figure 13.

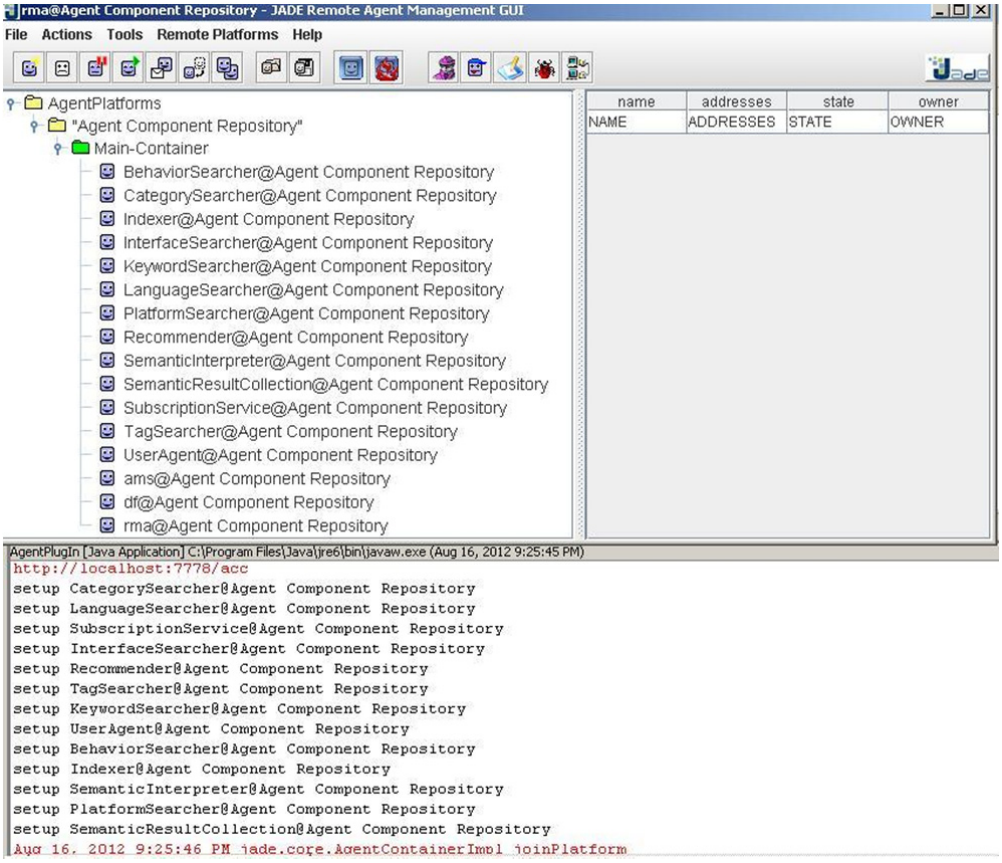

Figure 13: Agent Platform.

Figure 14 shows a screenshot of the home page with the existing categories in the repository and the agents associated with these categories.

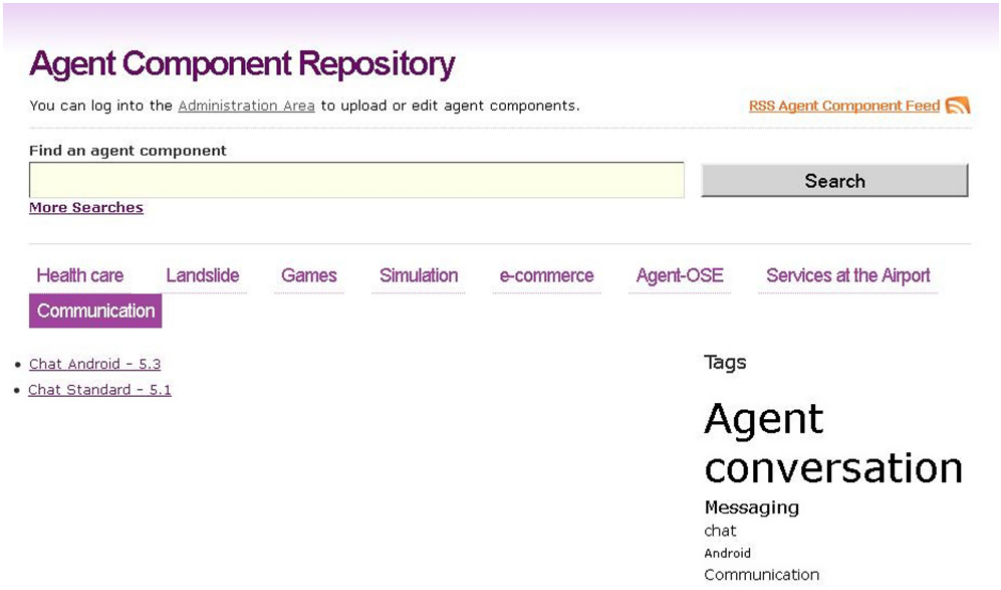

Figure 14: Home Page.

There are only two agent components classified in the category *Communication*. The union of their tags can be seen in the list of tags. The visualization of each tag depends on its weight in the system, that influences in the style of the font. If we select some tags, a filtered search is carried out pointing out which agents are assigned by these tags.

For example, for the specific category *Communication*, we click the tags *Agent Conversation* and *Messaging*, the list of agents associated with both tags, will be *Chat Android* and *Chat Standard*. But if *Android* is included to the filter, the result will be just *Chat Android* since *Chat Standard* was not registered with this tag. Figure 15 illustrates the current situation.

If we click a link that refers to a specific agent, we can see its characteristics. An example is depicted in Figure 16, which shows the description of the *Book Buyer Agent*, already aforementioned.

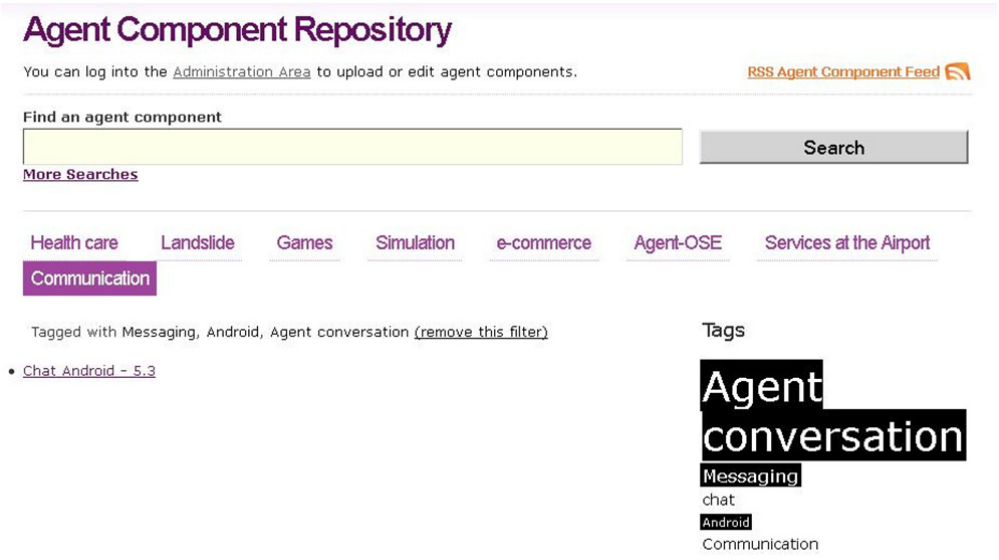

Figure 15: Tag Filtered Search

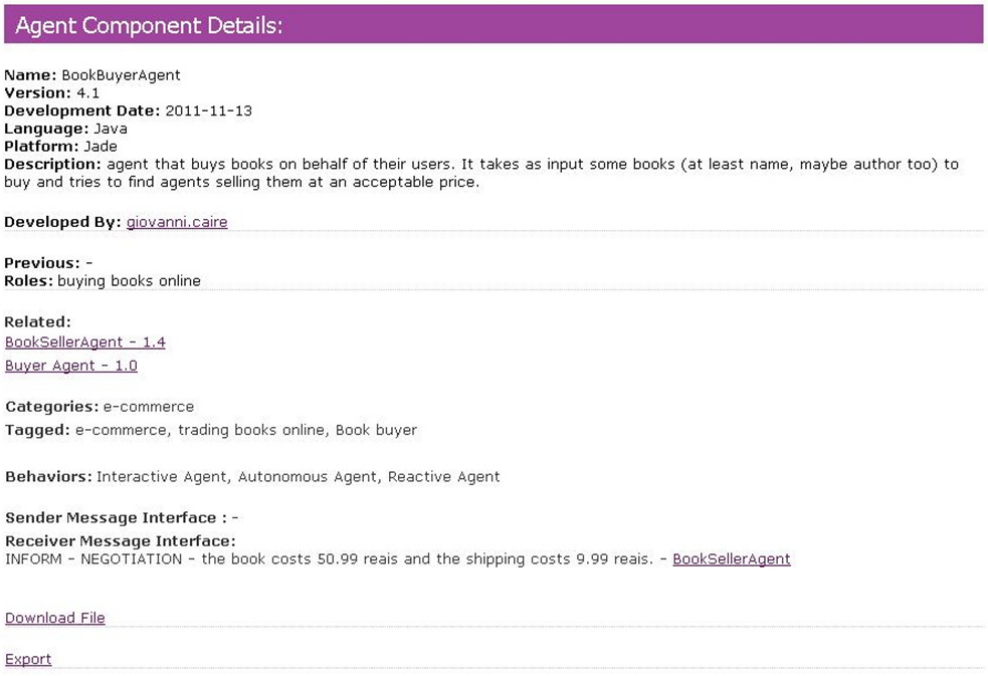

Figure 16: Agent's Description.

After a user logins in the system, he can register agent components how is illustrated in Figure 17.

Figure 17: Registering an Agent.

After the agent is registered in the repository, the next step is to add its interfaces and to know with which agents it interacts and how. Figure 18 shows how add the interfaces to a specific agent. The list of the agent participants was cut, not listing all the possible agents, to help with the visualization of the screenshot.

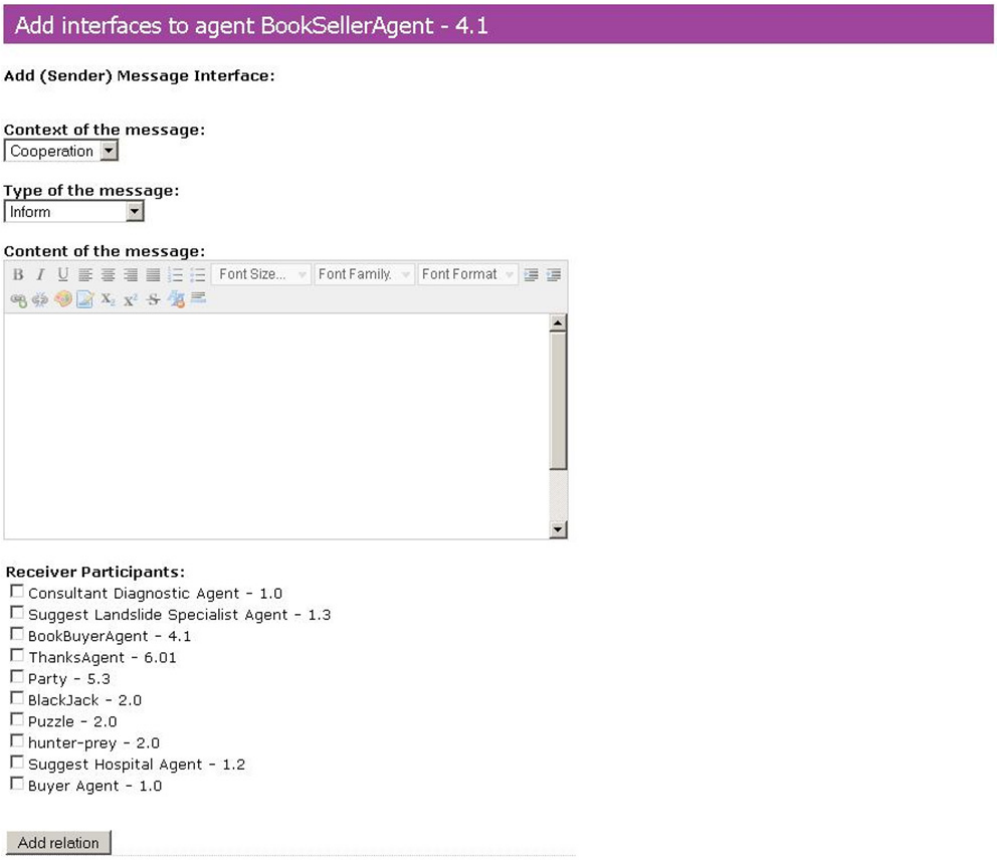

Figure 18: Adding Interfaces to the Agent.

The user can establish relationships among agent components, how it is illustrated in Figure 19. The list of the other agents was cut, not listing all the agents already stored in the repository, to help with the visualization of the screenshot.

To retrieve agents according to certain characteristics, the user can use a filtered search how is depicted in Figure 20.

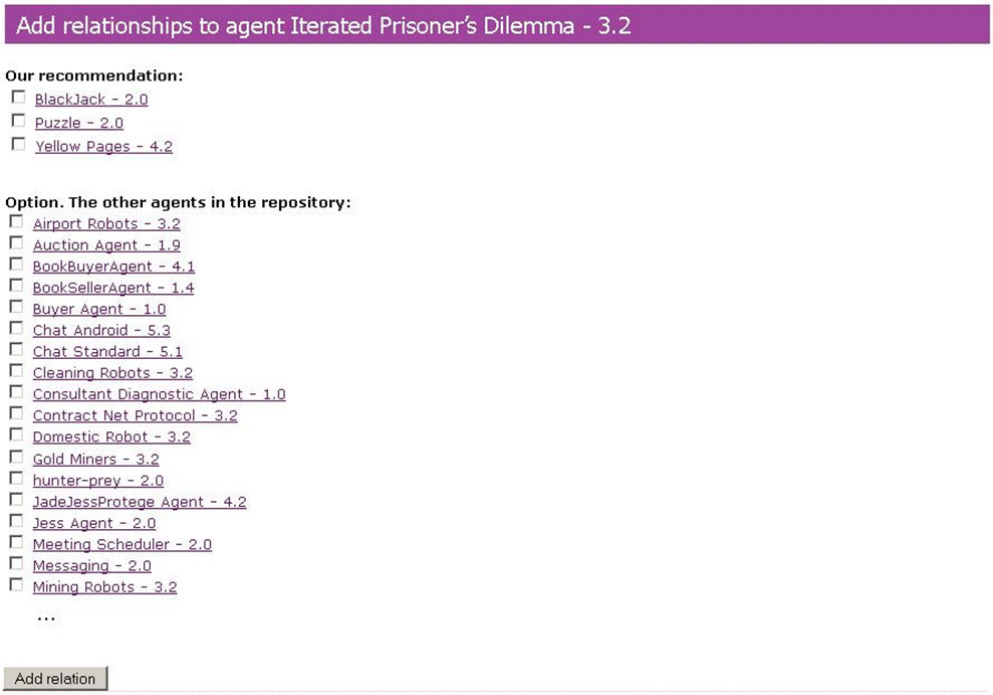

Figure 19: Adding Relationships to the Agent.

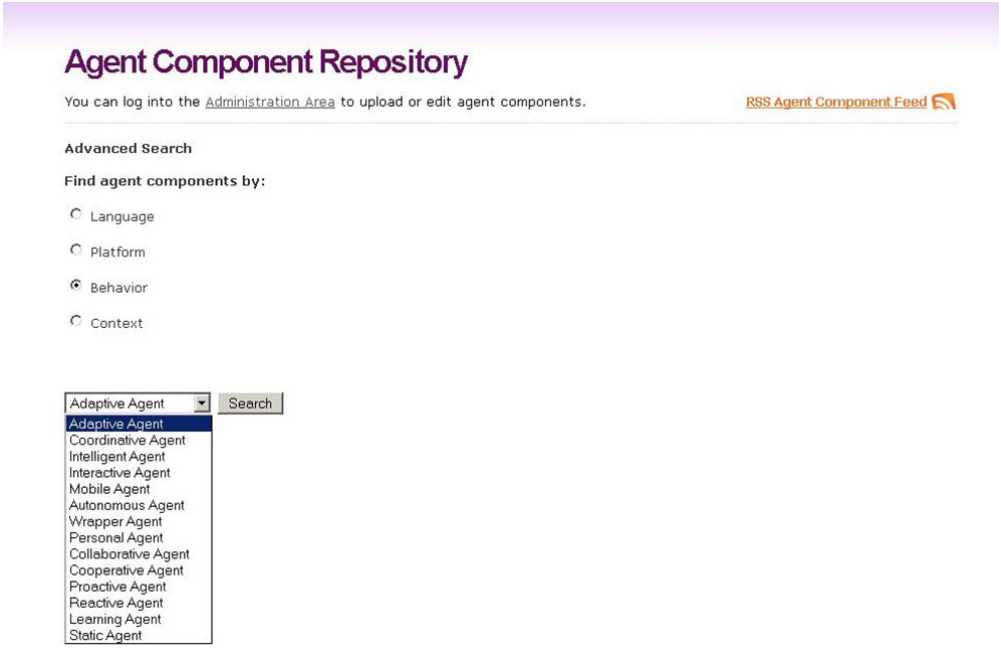

Figure 20: Other Search Methods.

A user can subscribe to some categories of agents to be updated about how it is happening in the current categories, as Figure 21 shows. There is a

description of each category and a description of each agent belonged to its category for users to be guided by. The available feeders are My Yahoo and Google, but the user can choose other applications, now built them into email clients and browsers.

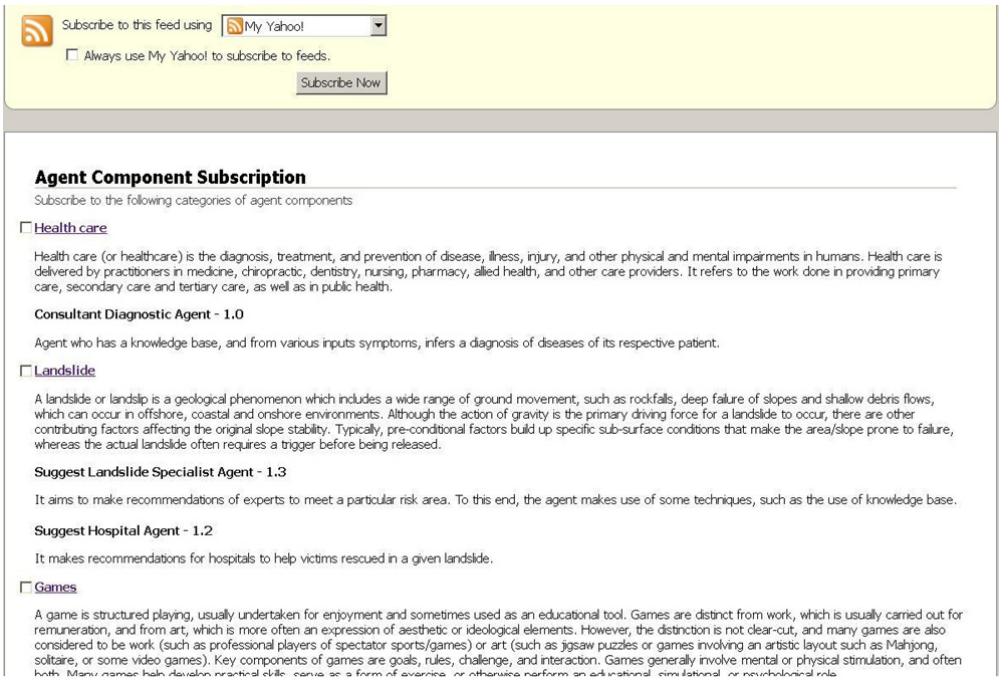

Figure 21: Subscription to Categories of Agents.

6.1. Implementation

The data collection of the repository was initially populated with the examples of software agents available in the web sites of JADE, Jadex and Jason [26] platforms, and also those developed by the Software Engineering Laboratory[5] (LES initials in Portuguese) at PUC-Rio[6]. All of these agents were implemented for different application domains.

The current version of the taxonomies consists of concepts of the agent components that populate the repository and the results of an online survey of

[5] http://www.les.inf.puc-rio.br/wiki/index.php/P%C3%A1gina_principal

[6] http://www.puc-rio.br/index.html

agent-oriented developers conducted at LES at PUC-Rio. The survey is showed in the appendix B.

The repository was implemented using modern technologies. To develop all the software agents in the system we adopted JADE framework, version 4.2, implemented in Java language. JADE simplifies the implementation while ensure interoperability of multi-agent systems due to it is compliance with the FIPA specifications, it means it not only assists the FIPA list of speech acts (label with the acts an agent can perform such as informing, asking and requesting), but also provides a middleware infrastructure that facilitates the agent communication (message exchange).

Since RDF is a W3C recommendation for representing metadata about web resources, we define the ontology that models the agents and the taxonomies to represent agent' behaviors and application domains, on RDF. These semantic concepts were developed with the version 4.1 of the free and open source ontology editor and knowledge base framework Protégé [64]. The Protégé platform, based on Java, provides a plug-and-play environment that makes it a flexible for rapid prototyping and application development. Ontologies in Protégé can be exported into a variety of formats including RDF, OWL, and XML Schema [83]. An important benefit of the ontology and taxonomies is that they are scalable, it means, new information can be added in them without affect the current versions.

The corpus of the agents' description was indexed by with Apache Lucene [48], version 3.6.1. Lucene is a free and open source powerful library for information retrieval with full text indexing and searching capabilities. It uses synonyms defined by WordNet [82], a large lexical database (a combination of a dictionary and thesaurus) for the English language. This database essentially represents word forms interchangeable lexically and semantically (such as synonyms, antonyms and homonyms) and its applications include search advertising, query expansion, document classification, and solving other languages processing problems.

We also used SPARQL [75] language to build unambiguous queries that search over the ontology and taxonomies. SPARQL was designed to express queries across diverse data sources, whether the data is natively stored as RDF or viewed as RDF via middleware. It is capable of querying graph patterns along

with their conjunctions and disjunctions, and also supports extensible value testing and constraining queries by source RDF graphs. To integrate SPARQL queries to the web application we employed Jena [27], version 2.6.4, an open source framework for building semantic web applications in Java. Jena includes an ontology API to handle OWL and RDF ontologies, a rule-based inference engine for reasoning with RDF and OWL data sources, a query engine compliant with SPARQL specification and a serve to allow RDF data to be published to other applications using a variety of protocols including SPARQL.

Finally, to develop the web application we used Play! Framework [60], version 1.2.4, which makes easy to build web applications with Java and Scala [70]. Play! follows the model-view-controller architectural pattern and is based on a lightweight, stateless, and features predictable and minimal resource consumption for highly-scalable application.