

Using Agents & Artifacts to Access the Semantic Web

A Recommender System Case Study

Keywords: Multi-Agent System, Agents & Artifacts, Semantic Web, Cartago, Recommender System, Learning Objects

Abstract: The integration of agents to the Semantic Web can significantly improve the Web services and the MAS systems, by increasing the knowledge available to agents to perform semantic tasks. In this paper we integrate the access of agents to the Semantic Web by means of the Agents & Artifacts model, using the Cartago framework. To test our approach, our case study is a recommender system to define metadata of a Learning Object according to a metadata standard. Our agents are able to provide recommendations to the user by querying the DBPedia through artifacts. Following the knowledge representation available on the Semantic Web, our artifacts aid the agents on the recommendations by querying linked data sources considering context-specific categories, classes or individuals. The main contribution of our work is to develop a prototype integrating MAS to the Semantic Web using artifacts.

1 INTRODUCTION

In this paper we research the use of the Agents & Artifacts (A&A) (Omicini et al., 2008a) model to ease the access of agents in a Multi-Agent Systems (MAS) to the functionalities of the Semantic Web. To test our approach we adopted the case study of the development of a recommender system for the educational area. The system is able to suggest values for metadata fields of Learning Objects (LO) based on partial knowledge provided by the user. The agents use the data to query the semantic repository DBPedia through specialized artifacts.

Multi-Agent Systems can be defined as a loosely coupled group of autonomous agents that interact in order to solve a problem (Jennings et al., 1998). The A&A model emerged from the need of modeling the environment of a MAS as a first-class entity (Ricci et al., 2007; Omicini et al., 2008b; Ricci et al., 2008; Weyns et al., 2007). Artifacts can model tools and resources to help the agents execute their tasks at runtime.

In this work the artifacts model the semantic searches to access the Web of Data (Berners-Lee et al., 2001), more specifically, DBPedia (Bizer et al., 2009). The Semantic Web is an extension of the current Web, its goals are to allow the effective sharing of data between large communities and to enable the understanding of this data by both humans and machines (Horrocks, 2008; Hendler and van Harmelen, 2008). Inside this context, the DBPedia project is a community effort to extract structured information from Wikipedia and to make this information accessible on

the Semantic Web (Bizer et al., 2009).

We access DBPedia to get recommendations for the metadata of a Learning Object (LO) through inferences on the partial knowledge provided by the user. A LO is any resource, usually a digital one, that may be used in a learning context (IEEE, 2002). Metadata provides additional information about a LO, like title, description, keywords, pedagogical information, authorship and rights.

The remaining sections of this article are structured as follows: We give an overview of the research background regarding A&A, Semantic Web and LOs in Section 2. Related work is presented in Section 3. Section 4 describes our approach to the access of agents to the Semantic Web and the system implementation. In Section 5 we discuss the results obtained in this work. Section 6 presents our conclusions and outlines future work.

2 BACKGROUND

2.1 Agents & Artifacts

In this work we use Jason to develop the MAS agents. Jason (Bordini and Hübner, 2007) is an Open Source interpreter for an extended version of AgentSpeak, a logic-based agent-oriented programming language. Given the grounding on AgentsSpeak(L), the agents in Jason follow the BDI model (Rao et al., 1995). Along with Jason, we adopt the Cartago (Ricci et al., 2009) infrastructure to model the environment as artifacts. Cartago is a platform and infrastructure that

provides a general programming model to build working areas where agents can use shared artifacts that represent the environment.

(Ricci et al., 2011) discusses two points of views regarding artifacts. From the agents' perspective artifacts are entities that model tools and resources, and are dynamically instantiated, observed, shared, grouped and used in order to help the agents to accomplish their tasks. From the programmer's point of view, artifacts are abstractions to model and implement a working environment that agents can use, so that they can interact with other agents and the external environment.

2.2 Semantic Web

According to the World Wide Web Consortium (W3C), the goal of the Semantic Web is to "allow data to be shared effectively by wider communities, and to be processed automatically by tools as well as manually." (Horrocks, 2008) states that to accomplish this, Semantic Web ontologies are used to provide extensible vocabularies of terms, each with a well-defined meaning. In computer science, an ontology models some aspect of the world by providing a vocabulary that describes various aspects of the domain being modelled and provides an explicit specification of the intended meaning of that vocabulary (Horrocks, 2008).

The need for an expressive and standard ontology language led to the development of the OWL ontology language standard. OWL uses an RDF-syntax and is based on Description Logics (Baader et al., 2003), a family of logic-based knowledge representation. It can represent rich and complex knowledge about things, groups of things, and relations between things. In addition, computer programs can process the knowledge expressed in OWL to generate logic inferences.

DBpedia (Bizer et al., 2009) is built upon the Semantic Web, its goal is to extract structured information from Wikipedia and to make this information accessible on the Web. Currently the DBpedia knowledge base describes over 2.6 million entities, along with their globally unique identifier, relationships, descriptions, classifications, etc. Those entities cover domains such as geographic information, people, companies, films, books, and scientific publications. DBpedia employs as its main ontology one that is a mapped version of the Wikipedia templates. Other ontologies are linked to DBpedia, such as Yago (Suchanek et al., 2007) and FOAF (Graves et al., 2007).

DBpedia provides a public SPARQL endpoint ¹ to query its knowledge base. The triple store OpenLink Virtuoso (Erling and Mikhailov, 2010) is used as the back-end database engine. SPARQL is the standard language for querying RDF data, it is based on graph patterns and subgraph matching (Hartig et al., 2009; Pérez et al., 2009). To query the SPARQL endpoint we adopted Jena API (McBride, 2002), an open source Java framework for building Semantic Web and Linked Data applications. DBpedia was chosen as our system knowledge base since it is based on Wikipedia, a source frequently used by students to obtain information. Furthermore, DBpedia is considered as one of the most famous basis of the Semantic Web (Berners-Lee, 2008).

2.3 Learning Objects

According to the IEEE (IEEE, 2002) a learning object is "any entity, digital or non-digital, which can be used, re-used or referenced during technology supported learning". Some examples are multimedia content, instructional content, learning objectives, instructional software and software tools. LOs can have additional information about them expressed through metadata. (Duval et al., 2002) defines metadata as structured data about other data, that can be objective (authors, creation data, version...) or subjective (keywords, abstract...).

A set of metadata can be grouped into a metadata standard, so that the metadata will have a well-defined semantic in accordance to the view of a specific community. The metadata standards for LOs favor the sharing, cataloguing, discovery and reuse of the learning objects (Duval et al., 2002). In this work we use the Brazilian standard OBAA (Bez et al., 2009). Our case study is based on the problem of specifying the metadata of a LO according to a metadata standard, so it can be easily shared and understood. This is a very tiresome task (Xavier and Gluz, 2009), which is related to the low production of LOs and frequent creation of LOs that don't follow a metadata standard.

3 Related Work

In this section we describe related work regarding the access of agents to the Semantic Web. At (Klapiscak and Bordini, 2009) the authors developed the agent architecture JASDL (Jason AgentSpeakDescription-Logic), an extension of the Jason agent platform. Their goal is to provide features of agent programming combined with ontological reasoning such as

¹<http://DBpedia.org/sparql>

plan trigger generalisation and belief base querying based on ontological knowledge. In order to implement this, the authors used the OWL-API as the DL-Reasoner, which they considered more elegant than the Jena API and also provided black-box debugging features. (Klapiscak and Bordini, 2009) considers JASDL as the first full implementation of an agent-oriented programming language with transparent use of ontologies and underlying ontological reasoning within a declarative setting.

At (da Silva and Vieira, 2007) is proposed Argonaut, a prototype that integrates Jason with the Semantic Web framework Jena to support context aware computing based on OWL ontologies. They developed a MAS for the mobile computing context, where the agents help the users to find out about locally situated services. Argonaut is the first practical approach integrating OWL, Jason and Jena (da Silva and Vieira, 2007). In their model all interactions with ontologies are defined within predefined internal actions in Java code. On the other hand, JASDL integrates agents to the Semantic Web in a more transparent way, improving the Jason belief base with the capacity for ontological reasoning.

At (Kagal et al., 2003) the authors describe ITTALKS, a web-based system that offers accesses to information about talks, seminars and colloquia related to information technology by the means of automatic and intelligent notifications. To accomplish this, the authors developed a MAS that access the Semantic Web using the DAML (DARPA Agent Markup Language) language. The DAML is employed to specify the ontologies used at the system, for knowledge base representation, reasoning, and agent communication. By using DAML, the agents at ITTALKS are able to infer extra information about the talks, such as+ finding talks compatible with the user's interests, schedule, location and current traffic.

Other relevant related works are as follows: (Zou et al., 2003) uses RDF and OWL as content and service description languages in the context of the framework TAGA (Travel Agent Game in Agentcities); (Chen et al., 2004) developed EasyMeeting, a pervasive smart meeting room system that makes use of agents and OWL ontologies to provide context-based services and information to meeting participants; and APL (Semantic Agent Programming Language) is introduced by ((Katsonov and Terziyan, 2008), an RDF-based middleware language for the Semantic Web that integrates the semantic description of the domain resources with the semantic prescription of the agents' behaviors.

4 The Recommender Model

In this paper we developed a recommender system for the metadata fields of a LO, based on an application profile of the metadata standard OBAA. To generate the recommendations our system uses a MAS composed of agents and artifacts specialized in accessing the DBpedia entities through SPARQL queries. Figure 1 shows an overview of the system. The user informs the metadata that she already knows to the system and then requests recommendations on other metadata fields, or to improve a metadata field already specified. Next, the GUI gathers the partial knowledge and sends it to the MAS, composed by agents and artifacts. Then the agents use the artifacts to query the DBpedia to find recommendations based on the partial knowledge initially provided. Finally, the recommendations are sent back to the GUI that shows them to the user.

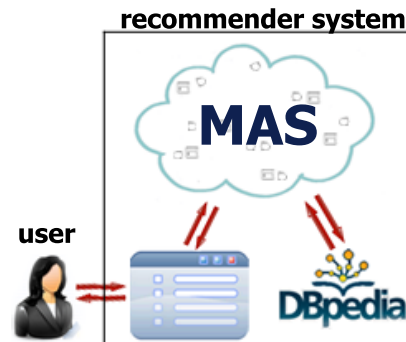


Figure 1: Overview of the system.

The chosen OBAA profile has four metadata categories: General, Lifecycle, Technical and Rights. We focused on the General category, more specifically on the metadata Title, Description and Keywords. We chose those three metadata because they are generic and so a wide range of recommendations can be generated for them. In addition, they can be easily mapped to some of DBpedia properties, such as `dbpedia-owl:abstract`, `rdfs:comment`, `db-prop:title`, `db-prop:name`, `foaf:name` and `rdfs:label`.

In Figure 2 is shown the architecture of the recommender model. The cloud represents the MAS, that contains the agents and artifacts. The InputArtifact and the OutputArtifact work as an integration between the MAS and the GUI, they also coordinate the agents. The InputArtifact provides the keywords to them and the OutputArtifact collects the results processed by the agents. The main part of the system is composed by nine agents and artifacts specialized in three types of semantic searches at DBpedia: individuals, classes and categories, where category is an

informational structure derived from Wikipedia. Each of these artifacts are mapped to one agent, so the system can process the queries in parallel, because the A&A model doesn't allow more than an agent using an artifact at a time.

The agents can access two operations at each artifact: the broad search and the narrow search. The broad search queries the DBPedia to obtain a large number of results in a short time. The narrow search obtains less results, but with a better quality. The agents access these operations by providing the keyword to be processed and the limit of the search. The artifacts return the results as the individuals that are more similar to the keyword. It is relevant to notice that all the three artifacts return individuals, in order to group and rank them. So after processing the query the classes and categories obtained by the respective approaches are processed again to get the individuals that belong to them.

Figure 3 shows an overview of the information exchange in the system, the diagram is in classical UML adapted to MAS. After partially informing at least one of the three metadata fields, the user can request the system to provide recommendations for one of the fields Title, Description or Keywords. The GUI gathers all the partial knowledge available, along with the fields from where they were obtained, and sends this information to the control layer.

The control layer is responsible for processing the partial knowledge in order to obtain keywords. Depending on what field an information originated from, it receives a different treatment. For the fields Title and Description the data is manipulated through Natural Language Processing (NLP), using the Apache Lucene library to remove plurals and stopwords (e.g.: he, she, my, does, was). The Keyword field is processed simply by separating the data into words by point and commas. The result is an array of keywords.

After acquiring all the keywords, they are ranked and the eight best ones are chosen and sent to the InputArtifact. By best we consider the keywords that showed up more times, so they probably are more important for the user. The selection of a few number of keywords is also important to reduce the processing time.

Every 0.5 seconds the agents check the InputArtifact to see if new keywords are available for processing. When keywords are available, each one is obtained by three agents, each of them specialized in a type of semantic search. The agents process the keywords in parallel through artifacts. After processing the keywords, the artifacts return ontology individuals. They are sent to the OutputArtifact that ranks them and selects the ten best individuals. Those in-

dividuals are used to obtain the values from properties that are compatible to the metadata originally requested for recommendations. Finally, the GUI gets the results from the OutputArtifact and shows them to the user, along with the URI of the individual from where the value was obtained.

4.1 Recommender System

In our model there are three types of artifacts, each one specialized in a kind of semantic search: by individuals, by classes and by categories. There are three kinds of agents, each one uses a type of artifact. Every artifact provides two ways of performing a semantic search: a broad and a narrow one. The broad semantic search is able to return more results and it is processed faster, but has a poor quality of results. The narrow search may be used on the results of the broad one, consuming an extra time but increasing the final quality.

The main difference between the broad and the narrow semantic search is which and how many filters are employed on the results and the depth used during the search. For example, the broad search for individuals filters the results by the classes they derive from. VideoGame, TelevisionShow and Band are some of the filtered classes from the DBPedia ontology. To further refine the results, the narrow search approach filters them based on what categories they belong, such as Sports, Companies and Comics. Our criteria to define the filters are based on if the class or category seemed to be relevant for an educational context.

In the individual approach to semantic search, the keywords are processed in order to obtain individuals that contain the keyword in the properties db-prop:title or db-prop:name. A shorter version of the SPARQL query for the narrow search of this approach is shown below. The words preceded by the symbol @ are parameters that are informed by the agents when calling the artifact's operation. The query starts by stating the prefixes, then it is defined what the query should return at SELECT and which criteria is used during the selection at WHERE. The filters ban the results that belong to prespecified categories. We also use the transitive option on the filters to limit the depth of the search. The parameters that the agent informs to perform the query are @keyword, the keyword being searched; @t.max, the depth of the search; and @limit, the number of results returned by the query.

```
PREFIX db-prop: <http://dbpedia.org/property/>
PREFIX dbpedia-owl:
    <http://dbpedia.org/ontology/>
PREFIX rdf:
```

```

    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:
    <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpedia_cat:
    <http://dbpedia.org/resource/Category:>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT distinct ?object
WHERE {
  {?object db-prop:title ?title . ?title
    <bif:contains> "+@keyword+" . }
UNION
  {?object db-prop:name ?name. ?name
    <bif:contains> "+@keyword+" . }
filter (NOT EXISTS
  {?object dcterms:subject ?Category .
    ?Category skos:broader dbpedia_cat:Sports
    option(TRANSITIVE, T_DISTINCT,
      t_max(@t_max)) }) .
filter (NOT EXISTS
  {?object dcterms:subject ?Category .
    ?Category skos:broader dbpedia_cat:Companies
    option(TRANSITIVE, T_DISTINCT,
      t_max(@t_max)) }) .
}
LIMIT @limit

```

For the classes approach the goal is to obtain the classes which have the property `rdfs:label` containing the keyword. Here the difference between the broad and the narrow search is just how many classes are being filtered. The parameters entered by the agents are again the keyword, the limit and the depth of the search. The results for the semantic searches are classes, so the agent calls an additional operation on the artifact to obtain the individuals belonging to those classes. The final results are individuals, instead of classes, in order to have a common ground between the results of the semantic searches, so they can be ranked and used to generate the recommendations. Below we show the SPARQL query used to obtain the individuals from a class. The prefixes used here are the same ones used on the previous SPARQL example. The query gets a quantity `@limit` of individuals that are subclass of a class until a depth of four.

```

select distinct ?subclass where {
?subclass a <"+"@class+"> option(TRANSITIVE,
  t_distinct, T_MAX(4))."
} LIMIT "+@limit;

```

Finally, the last approach is based on the DBPedia categories, that classify the ontology entities depending on their subject. Most categories are derived from Wikipedia and they are organized in hierarchies. This approach searches for categories whose property `rdfs:label` contains the keyword. The categories approach is similar to the classes one, where the differences between the broad and narrow searches are just the quantity of categories being filtered. Besides that,

after finishing the broad or narrow searches an extra operation is used to obtain the individuals that are classified under the resultant categories. We noticed that this search usually exceeded the memory limit for processing defined at the DBPedia server while filtering the results. To workaround it without losing quality of results, the search starts with a higher depth, 6, and decreases it by 2 until it reaches 0 or is executed successfully.

4.2 Agents and Artifacts Implementation

In this section we will describe how we implemented the recommender system according to the A&A model. When the system starts running the agent initializer instantiates all the artifacts for the MAS. The created artifacts are the `InputArtifact`, the `OutputArtifact` and a total of nine specialized artifacts, three for each of the three kinds of semantic search. Each type of artifact has three instances in order to improve the parallelism, since just one agent can use an operation in an artifact at a time in the A&A model. Besides the initializer agent, we create three agents for each type of search, a total of nine specialized agents. In order to the agents to know the names of the artifacts they are going to use, each artifact is instantiated with the same name of the agent that is going to use it.

Below we show a shorted version of the code for the `classes_approach` agent. It starts by discovering the artifacts that it will use through the goal `myTools`. Those are the artifacts for input, output and one specialized on the classes approach. Then the goal `consumeItems` checks if new keywords are available in the `InputArtifact` at 0.5 seconds intervals. When a keyword is obtained it is used in the `processItem` goal, which executes the semantic search. First, the agent calls the artifact's operation `execute_broad(Keyword, 30, R1)`, where 30 is the limit of classes that will be returned and R1 is the classes returned by the search. Following the initial operation, the agent uses the results from the broad search to execute a narrow search. If after executing the narrow search there are no results, the agent will send to the `OutputArtifact` the results from the broad search, otherwise the results from the narrow search will be sent. This decision is made by the goal `decideOutcome`. Before sending the results to the `OutputArtifact` through the `put` operation, the agent calls the operation `execute_getIndividuals(R1, 20, R3)` to obtain up to 20 individuals from each class as the final result from the semantic search. The agent code for the individuals and categories approach is similar to the one for the

classes approach. The difference for the individual approach is that the final result is already the output from the broad or the narrow searches.

```
!observe.

+!observe : true
  <- .wait(200);
  ?myTools(A1, A2, Id);
  !consumeItems(Id).

+!consumeItems(Id) : true
  <- .wait(500);
  get_for_ClassesApproach(Item);
  !processItem(Item, Id);
  !!consumeItems(Id).

(...)

+!processItem(Keyword, Id) : true
  <- execute_broad(Keyword, 30, R1)
    [artifact_id(Id)];
  execute_narrow(R1, R2) [artifact_id(Id)];
  isEmpty(R2, Test) [artifact_id("input")];
  !decideOutcome(R1, R2, Test, Id).

+!decideOutcome(R1, R2, Test, Id) : Test
  <- execute_getIndividuals(R1, 20, R3)
    [artifact_id(Id)];
  put(R3).

(...)

+?myTools(A1, A2, A3) : true
  <- lookupArtifact("input", A1);
  lookupArtifact("output", A2);
  .my_name(N);
  lookupArtifact(N, A3).

(...)
```

The operation for the broad search at the `ClassesApproach` artifact receives three parameters: the keyword, the limit of the broad search and a parameter that returns the result to the agent. Its execution starts by sending the SPARQL query, as explained in Section 4.1, to the DBPedia endpoint (<http://dbpedia.org/sparql>). Then the result is processed in order to obtain the classes resultant from the query. The Jena API is used to access the endpoint and process the query. The artifact operations for the other approaches and to get the individuals from a class or category works similarly. The difference is the SPARQL query being executed.

4.3 Results Processing

When all agents that obtained a keyword return the results to the `OutputArtifact`, the artifact ranks the individuals to get the ten best ones to generate recom-

mendations. Depending on the approach that generated the individual, it receives a different score. The total score for an individual is expressed below:

$$score = 1.3 * (indiv.) + 1 * (categ.) + 0.8 * (classes) \quad (1)$$

The individuals approach receives the higher score since our empirical tests indicate that its results had a better quality. One of the main reasons for this is because the filters are applied directly to the final results. We also noticed that the results from the categories approach were slightly better than the ones from the classes approach, probably because the categories at DBpedia have a stronger basis than the classes, since the categories are derived directly from the Wikipedia. The criteria to evaluate a result's quality is further explained in the next section.

After selecting the ten best individuals the `OutputArtifact` queries the DBPedia to obtain the values of their properties. The properties queried depend on what metadata field the user requested recommendations. Just one recommendation is returned per individual, but since there isn't a common property for all individuals the system tries to query different properties until a value is obtained. The priority list for querying properties for the `Description` field are `dbpedia-owl:abstract` and `rdfs:comment`. For the `Title` and `Keywords` metadata fields the queried properties are `db-prop:title`, `db-prop:name`, `foaf:name` and `rdfs:label`. We query the same properties for both `Title` and `Keywords` because we didn't find a property vastly used for keywords at DBPedia. In the case that more than one value is returned by a query, just the first one is taken as the recommendation for the individual. Finally, the list of the recommendations and the URI of individuals that generated them is shown to the user in the GUI.

4.4 Quality of Recommendations

To define the quality of the results we compared a set of individuals resultant from some keywords to a list of criteria items. The goal of this list was to have a general idea of the quality of the results. The list considered, for example, that if the individual is a subset or has a causal effect with the keyword, then it is a good result. If the individual is a company, sport team, holiday, book, or so on that just contains the keyword in its name, then it is classified as having a bad quality.

We noticed that the recommendations were useful to obtain more information about a concept or subtypes of a concept, but most of the time they had a bad quality for the educational context. The majority

of the concepts returned were unrelated to the keyword or too advanced. Another problem was that, due to the use of filters to try to improve the quality of the results, a semantic search could take between 1 to 10 minutes to process. We believe that these problems happened because DBPedia isn't a semantic repository specific for educational purposes, then lacking the pedagogical information necessary for the recommendations. So, a better quality of recommendations, and a better performance, could be obtained if we query a repository focused on the educational context or specific for LOs.

5 Results

In this paper we developed a recommender system where agents access the Semantic Web by means of artifacts that model the environment. The system is able to obtain recommendations for LO metadata using the partial knowledge provided by the user to process semantic queries on DBPedia through SPARQL. The contributions and results of this work can be summarized in three points of view:

a) Metadata recommenders systems: To the best of our knowledge, in the context of recommenders specific for educational LO metadata there are no works available to compare our results with. The system developed in this paper is an initial prototype for this context. In order to offer better recommendations for LO metadata, it would be necessary to evaluate our system accessing a semantic repository specific for the educational context.

b) Recommenders systems in general: In this work our goal was to develop a model that facilitates the access of agents to the Semantic Web, thus obtaining high quality recommendations was not our focus. Therefore we didn't evaluate our system focusing on recommendations, but we could do a proper evaluation in future work.

c) Architectures that enable agents to access the Semantic Web: The main objective of our paper was to develop a model of system where agents can access to the Semantic Web by means of artifacts, so this is the context where we are able to make most of our comparisons. In section 3 we presented related works where agents access the Semantic Web. The main difference between our work and the previous ones regarding the access to the Semantic Web by agents is that in this work the integration is accomplished by means of artifacts. In this system the artifacts are used as a middleware where the ontological interactions are processed, and additionally as coordinators of the agents.

The semantic artifacts use SPARQL queries with a predefined general structure. In the current version the parts customizable by the agents are the keywords being searched, the prefixes, the filters and some parameters of the search. In a future version we could use the functionality of internal properties of the artifacts. Internal properties can be modified at runtime by agents to change the services being offered by the artifact. So the agents would be able to customize the queries to DBPedia at runtime.

Due to the adoption of artifacts, our model provides reusability: By using the artifacts developed in this model, other agents are able to access the Semantic Web without the need to implement artifacts specific for them. It also isn't necessary to develop a whole new architecture or language in order to agents to access the Semantic Web. Nowadays Cartago, the framework that implements the A&A model, supports only the Jason agent language, but conceptually any BDI agent is able to use the artifacts developed in this work.

Other relevant points are that by using artifacts there is less computational burden on the agent side, since they don't need to waste time or resources processing the semantic query. The agents just activate the desired operation at the artifact and they are free to accomplish other goals while the artifact process the query. Finally, there is dynamic extensibility, a feature of the A&A model that enables agents to create and dispose artifacts by need at runtime. In (Ricci et al., 2011) the authors discuss in more details the advantages of the A&A model.

6 Conclusion and Future Work

In this paper we study the use of the Agents & Artifacts (A&A) model to ease the access of agents in a MAS to the functionalities of the Semantic Web, more specifically, of the DBPedia. Our research was tested with the case study of a recommender system for the metadata of Learning Objects (LO) based on the metadata standard OBAA. The agents query the DBPedia through operations available in specialized artifacts. We instantiate three types of artifacts, each one specific for a kind of semantic search: by individuals, by classes and by categories. In addition, the artifacts InputArtifact and OutputArtifact are used to integrate the GUI with the MAS, to compute pre and pos processing of the keywords and results, and to coordinate the agents.

The semantic artifacts developed in this work use SPARQL queries with a predefined structure, being quite restrictive. They could be reused without ma-

for changes to support other structures of SPARQL queries, but a total generic model would be hard to achieve because, at the end, the artifact's operations are static Java code. On the other hand, using Cartago artifacts provide reusability and interoperability: Any BDI agent can access the same artifacts, even if they were developed with different agent programming languages. Since each agent uses its own artifact, the semantic searches can be processed in parallel. Some other features of artifacts weren't used in this work, like the agents being able to configure the properties of an artifact to change its functionality at runtime, or creating and disposing artifacts at runtime to change the features of the environment.

Our main problem with using the A&A model was that it supports agent using various artifacts at the same time, but an artifact can't be used by more than one agent at a time. The A&A model was developed this way to avoid conflicts with more than one agent altering the internal state of an artifact, a feature that wasn't necessary for our approach. In our case, it was necessary to enable various agents to execute an operation to query the DBpedia at the same time to improve the parallelism. To solve this, we instantiated each semantic artifact one time for each agent in our model. The artifacts are instantiated with the name of the agent that will use them, so the agents would be able to know what artifact they should request. It was an effective approach for our case, but it isn't very elegant and may escalate quickly for a higher number of agents or semantic artifacts.

Possible future works for this project include: test the artifacts with other semantic repositories to try to obtain better recommendations; consider more metadata fields in the semantic searches; agents that monitor the GUI and make recommendations at real time; agents configuring the artifacts to modify the structure of the SPARQL queries at runtime; to utilize the contents of the LO or the user's personal data as context to process the semantic inferences; and to make an empirical evaluation of the system.

REFERENCES

- Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA.
- Berners-Lee, T. (2008). Sir tim berners-lee talks with talis about the semantic web. transcribed interview.
- Berners-Lee, T., Hendler, J., Lassila, O., et al. (2001). The semantic web. *Scientific american*, 284(5):28–37.
- Bez, M. R., Silva, J. M. C. d., Santos, E., Primo, T., and Bordinon, A. (2009). Obaa project: An approach to interoperable learning objects based on web and digital television. *Proceedings of the 9th WCCE*.
- Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009). Dbpedia-a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165.
- Bordini, R. and Hübner, J. F. (2007). A java-based interpreter for an extended version of agentspeak. *University of Durham, Universidade Regional de Blumenau*. (February 2007).
- Chen, H., Finin, T., Joshi, A., Kagal, L., Perich, F., and Chakraborty, D. (2004). Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, 8(6).
- da Silva, D. M. and Vieira, R. (2007). Argonaut: Integrating jason and jena for context aware computing based on owl ontologies. *Agent, Web Services, and Ontologies Integrated Methodologies*, page 19.
- Duval, E., Hodgins, W., Sutton, S., and Weibel, S. L. (2002). Metadata principles and practicalities. *D-lib Magazine*, 8(4):16.
- Erling, O. and Mikhailov, I. (2010). Virtuoso: Rdf support in a native rdbms. In *Semantic Web Information Management*, pages 501–519. Springer.
- Graves, M., Constabaris, A., and Brickley, D. (2007). FOAF: Connecting People on the Semantic Web. *Cataloging & classification quarterly*, 43(3):191–202.
- Hartig, O., Bizer, C., and Freytag, J.-C. (2009). Executing sparql queries over the web of linked data. In *The Semantic Web-ISWC 2009*, pages 293–309. Springer.
- Hendler, J. and van Harmelen, F. (2008). The semantic web: webizing knowledge representation. *Foundations of Artificial Intelligence*, 3:821–839.
- Horrocks, I. (2008). Ontologies and the semantic web. *Communications of the ACM*, 51(12):58–67.
- IEEE (2002). Ieee-sa standard 1484.12.1 draft standard for learning object metadata. Technical report, IEEE.
- Jennings, N. R., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38.
- Kagal, L., Perich, F., Chen, H., Tolia, S., Zou, Y., Finin, T., Joshi, A., Peng, Y., Cost, R. S., and Nicholas, C. (2003). Agents making sense of the semantic web. In *Innovative Concepts for Agent-Based Systems*, pages 417–433. Springer.

- Katasonov, A. and Terziyan, V. (2008). Semantic agent programming language (s-apl): A middleware platform for the semantic web. In *Proceedings of the 2008 IEEE International Conference on Semantic Computing, ICSC '08*, pages 504–511. IEEE Computer Society.
- Klapiscak, T. and Bordini, R. H. (2009). Jasdl: A practical programming approach combining agent and semantic web technologies. In *Declarative Agent Languages and Technologies VI*, pages 91–110. Springer.
- McBride, B. (2002). Jena: A semantic web toolkit. *Internet Computing, IEEE*, 6(6):55–59.
- Omicini, A., Ricci, A., and Viroli, M. (2008a). Artifacts in the a&a meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456.
- Omicini, A., Ricci, A., and Viroli, M. (2008b). Artifacts in the a&a meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456.
- Pérez, J., Arenas, M., and Gutierrez, C. (2009). Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):16.
- Rao, A. S., Georgeff, M. P., et al. (1995). Bdi agents: From theory to practice. In *ICMAS*, volume 95, pages 312–319.
- Ricci, A., Piunti, M., and Viroli, M. (2011). Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192.
- Ricci, A., Piunti, M., Viroli, M., and Omicini, A. (2009). Environment programming in cartago. In *Multi-Agent Programming*, pages 259–288. Springer.
- Ricci, A., Viroli, M., and Omicini, A. (2007). Give agents their artifacts: the a&a approach for engineering working environments in mas. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 150. ACM.
- Ricci, A., Viroli, M., and Omicini, A. (2008). The a&a programming model and technology for developing agent environments in mas. In *Programming multi-agent systems*, pages 89–106. Springer.
- Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM.
- Weyns, D., Omicini, A., and Odell, J. (2007). Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, 14(1):5–30.
- Xavier, A. C. and Gluz, J. C. (2009). Relatório técnico rt-obao-06 subgrupo agentes e ontologias: Análise comparativa de editores de objetos de aprendizagem. Technical report, UFRGS.
- Zou, Y., Finin, T., Ding, L., Chen, H., and Pan, R. (2003). Using semantic web technology in multi-agent systems: a case study in the taga trading agent environment. In *Proceedings of the 5th international conference on Electronic commerce*, pages 95–101. ACM.

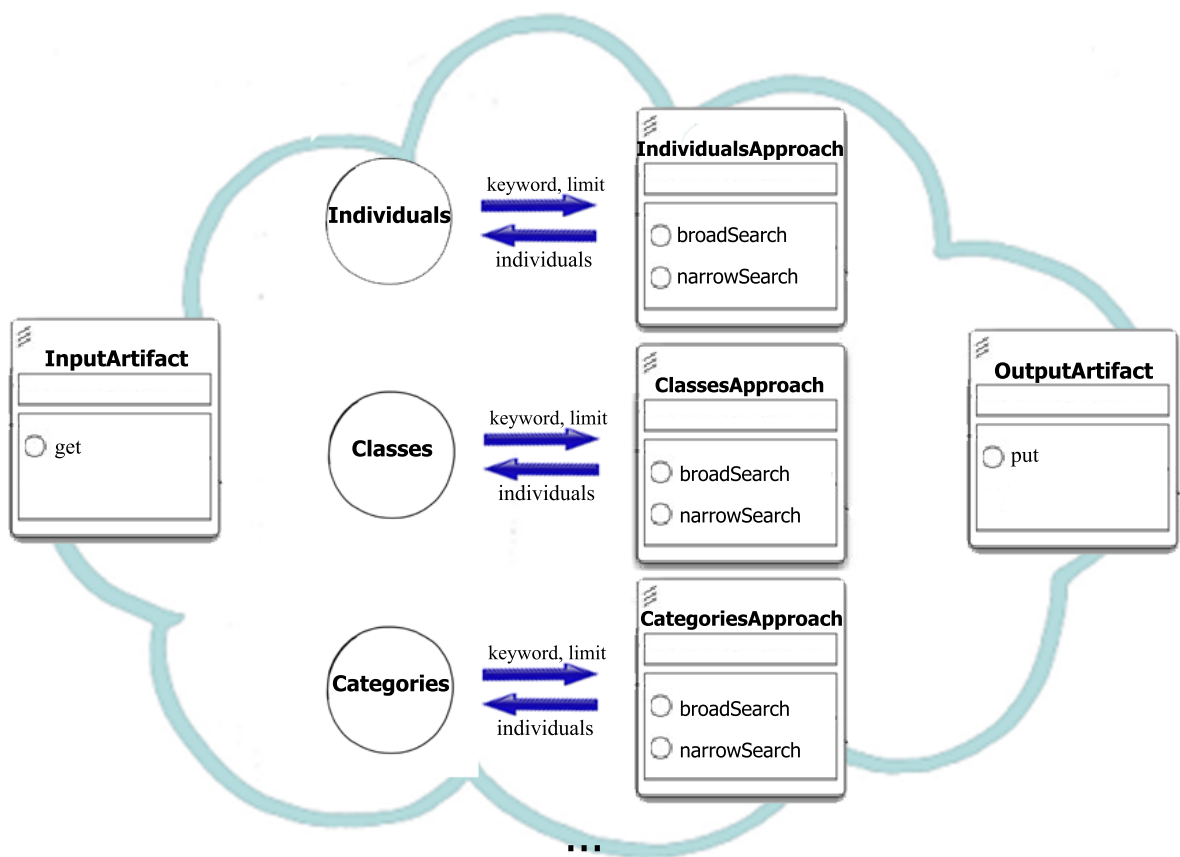


Figure 2: Overview of the recommender model.

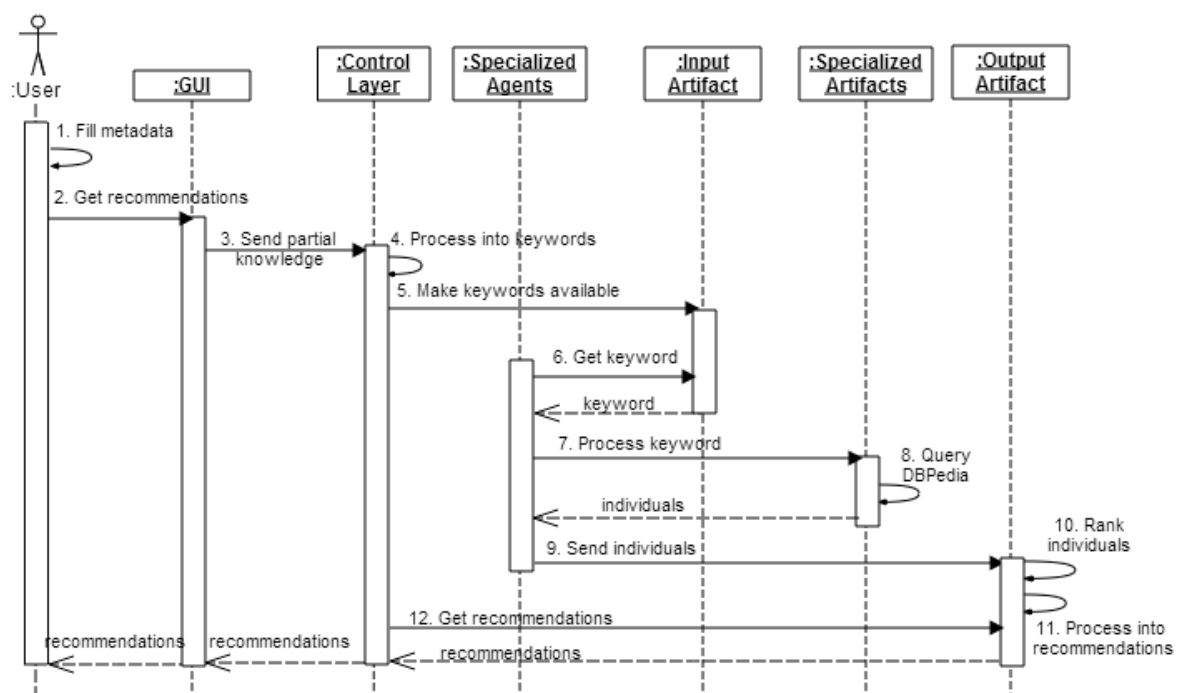


Figure 3: Sequence Diagram with an overview of the information exchange in the MAS.