Abstract—We posit that a robust development environment for
the construction of agent-oriented software systems must be
enhanced by advanced reuse methods. However, research
addressing agent reuse is meager and does not tackle the
problem of organizing and storing agent-oriented artifacts according
to the software engineers' needs. Therefore, the agent retrieval
process turns into an important challenge to be overcome in
agent-oriented software engineering. In this context, this paper
proposes a repository prototype based on semantic web
technologies that support reuse for developing agent systems. The
repository includes: (1) a meta-model for representing the agents,
represented by means of an ontology, (2) a taxonomy to classify
agents according their application domains, (3) a
recommendation system that allows end-users to discover
reusable interrelated agents, and (4) enhanced search and
browsing methods for agents. Finally, we discuss an illustrative
example that shows the proposed reuse method is an
improvement in terms of the relevance of retrieved agent-
oriented artifacts.

Software agent reuse; agent component; agent-oriented
artifact; agent repository; semantic information retrieval; agent-
oriented software engineering

I. INTRODUCTION

Nowadays the development of complex systems is
extremely common. Considering this context, the multi-agent
system (MAS) paradigm [1] has been used especially when
distributed, autonomous and pro-active entities are represented.

A MAS can be composed of several software agents [1],
which interact among them to achieve common goals.
Developing an agent-based system from the ground up is a
difficult, expensive, and lengthy software-engineering activity
due to the various kinds of expertise necessary. In developing a
multi-agent system, it is necessary to define multiple agents
that communicate and cooperate among themselves while
engaging in group decision making as well as competitive
behavior. The benefits of applying software-engineering
principles to guide problem solving have not been fully
appreciated and applied by the agent-development community.

According to [2], reusing agents created previously for
other systems is a crucial point that must be considered in the
agent-oriented software engineering (AOSE) [3]. It brings
several advantages to multi-agent systems such as: (1)
construction of reliable and consistent software since it uses
code created and tested, (2) reduction of cost and time
developing agents, and (3) increased flexibility of the software
to facilitate its maintenance and evolution.

Today, although software artifact reuse is already
established in the literature on software engineering, the work
addressing agent reuse is meager and does not tackle the
problem of identifying, organizing and storing agent-oriented
artifacts for reuse. Hence, the process of retrieving existing
reusable agent-oriented artifacts from different application
domains, or agent retrieval, is limited by the absence of
appropriate mechanisms and standards. In this context, the
entire agent retrieval process, which includes indexation,
identification, storage and recovery, turns into a crucial
impediment to be overcome in AOSE.

In order to provide a solution, we propose a repository
prototype based on semantic web technologies that identifies
and locates appropriate reusable agent components according
to user needs. A (software) agent component is a software
component [4] that takes advantage of both software agent and
component techniques (reusability, economy, extensibility).
The interface of an agent component is part of a component
that defines explicitly and semantically its access points (e.g.,
communication protocol), which make the agent services
available to the environment and accessible by other agents.

For a software agent to be effectively reused, its
specifications must be flexible enough and easy to adapt to the
many variations that can exist in an application domain.
Consequently, we created an ontology to model the agent
components based on characteristics stipulated by the
Foundation of Intelligent Physical Agents (FIPA) [5].
Ontologies have emerged as a powerful tool to enable
knowledge representation and knowledge sharing, by a
community of a target domain in an explicit and formal way,
and to achieve semantic interoperability among heterogeneous
distributed systems [6]. Thus, ontologies enable knowledge
reuse and a standardized model of software artifacts.

II. RELATED WORK

There have been some initiatives on exploiting software
reuse in agent technology [7][8]. The works [2][4] discuss the
need of toolkits and libraries of agents, agent parts or pre-
connected agent societies, that could allow their use in different
systems. Researchers have explored the possibility of using
agents as advanced reusable artifacts of software. In this
scenario, agents should exhibit at least the same properties of

Semantic-based Repository of Agent Components

Merlin Parra Jiménez, Andrew Diniz da Costa, Carlos José Pereira de Lucena
Software Engineering Laboratory
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Brazil
merlin@les.inf.puc-rio.br
{acosta, lucena}@inf.puc-rio.br
software components, adding interesting characteristics like autonomy, reasoning and goal-direct behavior [9][2].

We realized which difficulties exist to reusing software agents, such as the lack of mechanism or standards to represent and retrieve agent-oriented artifacts, and the lack of architectures and programming languages that allow MASs to implement all the features of agents. The capabilities of existing component repositories are not sufficient to provide for the reuse of heterogeneous agent-oriented artifacts, since they do not support a model of agents that includes their dimensions and interactions, or their retrieval according agent characteristics. Therefore, we propose a repository prototype based on semantic web technologies that exploits reuse in agent-oriented development among different agent architectures, platforms and languages in diverse application domains.

III. REPOSITORY OF AGENT COMPONENTS

The repository provides a web interface that implements the searching and browsing mechanisms to allow the user to view which agents offer certain functions in a particular category or with particular specification, furthermore submit search queries. The feature of searching artifacts is supported by a recommendation system.

In order to register an agent component, a description of the agent has to be provided by the user. Users can edit only the agent components added by them. Users can download an originally created agent component from the repository, a newer version of existing component or its entire predecessor’s history. These functionalities are supported by the core of the repository, whose main characteristics are described next.

A. Modeling Agent Components

Although agents are developed under diverse architectures, platforms, languages, and agent components are different in their interfaces, internal architecture and functionalities, we observed they could be modeled within a standard and intelligible representation.

We defined a meta-model to translate the dissimilar agent components into this representation. It establishes a precise and formal description that shares the understanding of agent components regardless of the types of components that are in the repository. Therefore, heterogeneous agent components can be stored and retrieved without losing any of their own characteristics.

For structuring the meta-data with the information of agents provided by the user, the repository uses an ontology. An important advantage of the ontology to maximize the reusability is to allow the extensibility of agents, i.e., to extend features of an agent (e.g. collaboration protocols or knowledge) without breaking the previous architecture or the reusability of the agent in the system that contains it. Thus, the ontology is open for anyone to use and explore.

After a thorough examination of available research on ontologies that describe software agents (functionalties, structure and interfaces), no appropriate result for modeling the common structure of heterogeneous agent components was found. Hence, we complement existing ontologies since some agent characteristics are not still covered, for example the interfaces of heterogeneous agent components. Therefore, we propose the ontology depicted in Figure 1 to formalize the description of dissimilar agent components.

A reusable agent component should have a number of attributes that are essential in determining the appropriateness of this construct in the reuse process. These attributes, which are considered as meta-data for the agent component specification are the following:

a) ID: Identifier given to the agent. It is unique in the system.

b) Name: Name given to the agent.

c) Description: Description of the agent in natural language, which can refer to whatever information related to the agent.

d) Version: Development version of the agent.

e) Previous Version: If the version is not the first one, the agent is a new one based on a previous version.

f) Date: Date when the agent was developed.

g) Language: Programming language the agent was developed with.

h) Platform: Platform the agent was developed with.

i) User: Developer or team responsible for implementing the agent.

j) Dimensions: Characteristics of the agent according to its nature, like autonomy, reactivity, etc.

k) Roles: Roles the agent performs.

l) Operations: Operations each role executes.

m) Parameters: Requirements an operation has to execute.

n) Categories: Application domains associated to kind of roles.

o) Tags: Tags the agent can be described with a few of words. Each tag has its own weight in the system.

p) File: File (JAR or ZIP for example) which contains the agent component.

q) Related: There are agents related to the current one respect to some characteristics (description, roles, previous version, interfaces, categories and tags).

r) Interfaces: Unlike software components, there is not a standard format to document agents’ interfaces based on the interfaces of public functions that include restrictions on the behavior of objects, such as the order in which the functions/operations should be invoked. To support communication, compatibility or interoperability among all of these heterogeneous agents, we propose an interface model that describes the context (cooperation, coordination, and negotiation) of relations among the agents, the types of messages the agent sends or receives based on the communicate acts of FIPA [5], the content of the message that can include ontologies, and the other agent participants.

s) Diagram: Image file to illustrate the structure of the agent showing its class and its relationships in its environment.

t) Documentation: URL or plain text that explains how an agent interacts in its environment or how it should be used.
An individual example of this ontology is depicted in [10]. It consists on an agent that buys a specific book at the lower price at online seller libraries.

B. Classifying Agent Components

Classifying agent-based artifacts allows software engineers to organize collections of them into structures that they can look easily for in future searches. In order to perform such classification, a hierarchical taxonomy of application domains was adopted. However, there are several perspectives to classify existing software agents, which are not unique.

The taxonomy facilitates to the user during a search, to access not only the agents related to keywords in the query, but also those agents that are interrelated semantically to them, such as those with equivalent characteristics or terms. It reduces the search space and makes the search results more relevant. In [10] there is a graphical subset of our taxonomy and the complete version in owl format. It is built with several facets that compose the agents’ domains, using a common vocabulary, derived and defined by the agent developers/users.

C. Recommendation System

To support the search in the repository, we created a recommendation system (RS) that allows end-users to discover the existence of reusable interrelated agents, and to learn new information about agents as needed, improving user productivity and promoting agent reuse. To measure the degree of similarity or relationship among the artifacts in the repository, the RS adapts the tf-idf algorithm [11], to compute weights of terms belonging to the agent components, and advises the user which agents would be related. Later, the user can establish the relations, and a degree of relationship is assigned automatically. The RS tracks usage histories of a group of agents to recommend agents expected being needed by that user. In that way, an ontology network, that contains the discovered data and their associative relations, is constructed.

The attributes taken into account for the recommendation are the description, roles, interfaces, ancestor, categories and tags.

D. Semantic-based Search System

For retrieving suitable agent-based artifacts, we developed a suite of search methods that utilizes the semantics of the agent descriptions, streamlining and reducing the search time. The whole search process combines these phases: (1) pre-processing the agent’s specifications and queries i.e., conflating related words to a common word stem, (2) interpreting user queries to choose the specific domains and establish the search method and parameters in terms of the characteristics of agents, (3) splitting these domains into sub-queries and executing them, (4) retrieving the agent components, rating and filtering them according to their semantic relevance with user parameters, (5) analyzing and classifying the results to infer if there is new relevant information that can enrich the semantic knowledge base, and (6) presenting the results to the user in the way further exploration of each component is enabled. The semantic knowledge base contains the data from the ontology and taxonomy, the indexes, the index terms and the learned and inferred knowledge during the search process. The indexation is performed by means of a lexical database of English.
The different ways to retrieve agents in the repository are described below.

1) **Keyword-based Searching:** Initially, we create a vector for the query with its terms. Afterwards, we calculate the cosine similarities [11] between the query vector and the vectors that represent the index terms of all agent components already stored in the repository.

2) **Tag-based Searching (Tag Cloud):** It is a weighted list that exemplifies the density of keywords present according to its relevance on the agent characteristics using a variety of fonts in the visual design. This allows a user to quickly identify what archetypes of agents are more common in the repository. Selecting a tag, the tag cloud will work as a browser leading to a collection of agent components that are associated with that tag.

3) **Custom Search Facet:** A tree-structure-based search representation model is conceived to allow users to browse, locate and filter their search intention by functionality. After a user lists all agent components related to an application domain, he can refine that result set. For this, all tags of all resulting agents are showed and the user under that category can mark the tags that he is interested on. Thus, the agent component result set is refined.

4) **Platform-based Searching:** It is looked up on the ontology all agent components already stored in the repository that were developed in the platform passed as parameter. It is an analogous process respect to the programming language.

5) **Interface-based Searching:** It is looked up all the agents that interact in a certain context, e.g. cooperation, coordination and negotiation, to achieve a specific role. In addition, it can be looked up the specification of message exchange or the description of the participants to accomplish this interaction.

There is a user manual in [10] that contains all essential information for the user to make full use of the repository system.

IV. **ILLUSTRATIVE EXAMPLE**

To evaluate correctness (task-performance) and effectiveness (time-performance) of a method of retrieval software artifacts in the repository, we rely on two metrics that are standard practice for retrieval, recall and precision [11]. Recall means that all the relevant components, while precision means that all the retrieved components exactly match the query submitted by a user.

To evaluate our information model, we developed a study that involves twenty-eight agent components stored in the repository [10] and a produced test collection. A test collection is a set of elaborated queries that cover all the components and the associated set of relevant components that is known a priori. These artifacts were implemented using Java language but with dissimilar platforms in different application domains by unrelated developers.

Then, the search of each query is performed using mainly the keyword-based, tag-based and interface-based approaches, with a fixed minimum relevance acceptance value of 0.6 [11]. Since we know a priori which agents are relevant among all that are returned to that query and how many relevant artifacts are in the repository, we can calculate the recall. We also know the quantity of agents retrieved, making it possible to calculate the precision.

We realized that the values of recall and precision have the same behavior in the case of the searches based on language, platform, categories and tags. Just in this particular case, an exact match of the query and the respective attribute was made.

To conclude, we find that a semantic approach can significantly improve precision and recall of search methods.

V. **FINAL REMARKS**

This paper presents a prototype repository to support reuse for developing agent-oriented systems. The repository relies on semantic web technologies for the description and retrieval of the components: (1) an ontology to set up all the characteristics associated with heterogeneous software agents, and (2) a taxonomy to classify these components according to their application domains.

We aim (1) to implement some combination of the search techniques. For instance, we could combine a keyword-based approach with the facet-based search or consider combinations of attributes as input data, e.g. fixed platform and programming language; (2) to represent the (semantic) intersection among application domains with the taxonomy. We will have to analyze if both of these improve the relevance of the searches. In addition, we must evaluate the performance of the search system taking into consideration the continual agent component registration, since new concepts can be included dynamically in the taxonomy and the tag cloud.

REFERENCES