Implementation of Agent-based Ontology Mapping and Integration

Li Li, Yun Yang and Baolin Wu
Faculty of Information and Communication Technologies
Swinburne University of Technology
PO Box 218, Hawthorn, Melbourne, VIC 3122, Australia
{lli, yyang, bwu}@ict.swin.edu.au

Abstract

The achievement of some business goals requires more than individual capabilities and knowledge. Hence, it is necessary for different organisations to work together in order to achieve the goals. In this paper, an innovative agent-based architecture is developed to model ontology mapping and integration in a distributed and dynamic business environment where interactions among organisations may take place at unpredictable times, for unpredictable reasons, and between unpredictable organisations. Functionalities and implementation of involved agents are discussed. In addition, a three-layer user ontologies are introduced. Finally, demonstrations of ontology mapping and integration are presented.

1 Introduction

Ontologies facilitate the interoperability between heterogeneous systems involved in commonly interested domain applications by providing a shared understanding of domain problems and a formalisation that makes ontologies machine-processable. The benefits of using ontologies have been recognised in many areas such as knowledge and content management, and e-commerce. The proliferation of Internet technology and globalisation of business environments has given rise to the advent of an ever-surprising varieties of ontologies which are far beyond expectations, whilst the changeable environment enforces underlying ontologies evolving over time. Moreover, interactions among organisations may take place at unpredictable times, for unpredictable reasons, between unpredictable organisations. Agents in multi-agent systems (MAS) operate flexibly and rationally in an environment which are dynamic and heterogeneous, given that agents have abilities to perceive changes of environments and respond in a timely fashion.

Of the architectures of modelling ontology managements (including ontology mapping and integration), most of the existing work attempt to build systems by using specific techniques to define similarity measurement, or to solve problems such as matching in ontology mapping [5] rather than focus on architectures. For instance, PROMPT [7] suggests to designers which concepts may be related. Only little of existing systems or tools claim that they have considered architectures and supporting techniques. In which InfoSleuth [2], an agent-based system, is worth noting. It is implemented in Java, and the agent communication language is KQML [1]. However, as stated in its future work, it lacks important capabilities such as managing agent systems as provided in some current agent software. Moreover, it is not FIPA-compliant and has no intention to create mapping automatically in the proposed system.

Although the problem of specifying the architecture is at the core of ontology mapping and integration on the Web, it has not been thoroughly investigated yet. To achieve the flexibility, agent technology finds its niche in operating over heterogeneous ontologies in a dynamic and distributed environment where substantial support for change is necessary.

This paper is organised as follows. Section 2 introduces an overall agent-based architecture. Section 3 discusses ontologies in this work. Section 4 describes agent designs and implementations. Section 5 demonstrates ontology mapping and integration. And finally, Section 6 is the conclusion.

2 Agent-based Architecture

Agent technology presents an exciting prospect in a field where high dynamics are required. It has the potential to significantly extend the range of applications that can feasibly be tackled [4].

2.1 Rationale

The emerging Web technology has made the environment of ontologies and ontology-based applications change

IEEE International Conference on e-Business Engineering (ICEBE'06)
0-7695-2645-4/06 $20.00 © 2006 IEEE

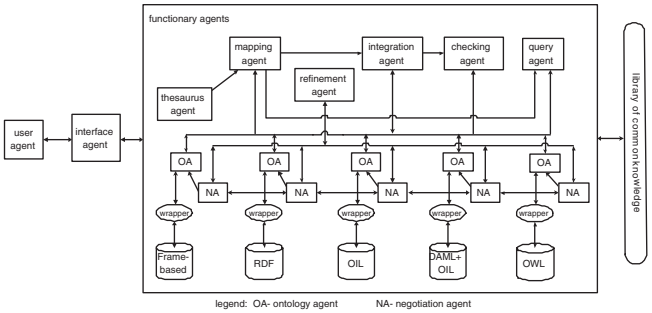

Figure 1. Agent-based framework

rapidly, which also has made ontology integration more difficult than ever before. The characteristics of the environment are as follows:

- Available ontologies are scattered on the Web which make manually gathering of relevant information for a particular problem nearly impossible;
- The number of organisations (with different ontologies) may change dynamically by organisations entering or leaving the environment randomly. Furthermore, which ontology is available along with a particular organisation is uncertain;
- Ontologies are prone to adaptation and evolution;
- Processes of different organisations operate concurrently to modify the environment in ways that an organisation has no control over;
- Different organisations demand efficient interaction as these interactions may occur at unpredictable times, for unpredictable reasons, between unpredictable organisations. An organisation thus needs to consider *synchronising* or *coordinating* its actions with those of other processes in the environment.

MAS's abilities of providing robustness and efficiency and solving problems in which data or control is distributed, make the MAS perfectly suitable for dynamic, distributed environments. Agent technology is thus ideally suited to model ontology mapping and integration on the Web.

2.2 Agent-based Architecture Overview

The overall framework of the agent-based ontology management is shown in Figure 1 [6]. The functionalities of each agents in this environment are briefly described below with details are given in Section 4.

User Agent (*UA*): This agent interacts with a particular GUI. It includes getting the business scenario from GUI, passing it on to the *interface agent* (*IA*) and presenting expected results.

Interface Agent (*IA*): This agent interacts with the *UA*. It is in charge of assigning a particular task to one or several agent(s) of a virtual community[1].

Ontology Agent (*OA*): This agent acts on behalf of a certain ontology. It behaves properly in a specified agent platform. It is equipped with functionalities of a certain ontology (e.g. it operates over the ontology structure and a particular intermediate result structure). The main purpose of the *OA* is to perform ontology related tasks which are isolated from external *functionary agent*s. The presence of *OA* allows flexible system organisation.

Negotiation Agent (*NA*): This agent takes part in negotiation setting in an attempt to obtain ontology evolving information for corresponding *OA*.

Functionary Agent: It consists of agents providing functionalities of thesaurus similarity, ontology mapping, ontology integration and consistency checking.

- **Thesaurus Similarity Agent** (*SA*): This is in charge of maintaining thesaurus similarity within a suitable structure. The major tasks of the *SA* are to: (1) append new concepts to the specified structure; and (2) search a particular concept in the structure.
- **Mapping Agent** (*MA*): This is shaped to provide linkages to pave the way for the interoperability of heterogeneity of various ontologies on the Web. It is the foundation of further ontology integration. The major task of the *MA* is to estimate whether two given concepts mapping each other according to its knowledge.
- **Integration Agent** (*InA*): It is responsible for ontology integration based on a certain business scenario. The major tasks of an *InA* are to: (1) count the appearance of each specified concept; and (2) filter the unexpected items before sending them to the *OA*.
- **Refinement Agent** (*RA*): This maintains ontology coherence and integrity in an environment of dynamic changes of ontologies that may take place frequently. The major tasks of a *RA* are to: (1) obtain up-to-date information for a specified concept; and (2) locate any differences between a previous description and the current one.
- **Consistency Checking Agent** (*CA*): This checks the consistency of an ontology. The major tasks of a *CA* are to: (1) check if a particular concept is the subclass of two disjoint concepts; and (2) warn the *UA* if inconsistency exists.

[1] A group of partners commonly interested in a certain business scenario.

- **Query Agent** (*QA*): This is in charge of a query based on created ontology mappings between pairs of ontologies. It maintains the query module in a distributed environment.

Library of Common Knowledge: This library includes atomic roles and primitive concepts which comprise the baseline of knowledge in a particular domain. It may initially be created or become created during run time through agent communications.

Wrapper programs, attached to each ontology sources, handle ontology formatting transformations between the local representation model and the internal representation model of the proposed system.

2.3 Agent Communication Languages

Agent communication language is FIPA ACL. In essence, an agent communication language provides a set of communication acts for agents in a MAS to perform. The purpose of agent communication is to convey information about an agent's own mental state with the objective of affecting the mental state of the communication partner. A three-layer model of FIPA ACL (*http://www.fipa.org/repository/aclspecs.html*) includes the *content* (layer) of the message; the *message* (layer) of particular attitude towards the content in forms of performatives of ACL [9]; and the *communication* (layer) of mechanics of communication. The concrete syntax for FIPA ACL messages closely resembles that of KQML [1], but the ACL language has greatly enhanced the semantics of passing messages.

Agent interaction will be demonstrated in AUML [8] throughout this paper. In the following sections, AUML diagrams will be introduced to describe interactions among agents for a particular task.

3 Ontologies

In this paper, agents perform ontology mapping and integration related work rather than just consume ontologies as usual. We develop a three-layer model (Figure 2) to represent ontologies in this system. This model accommodates different ontology languages and representations when instantiating various ontologies. The three layers are: E-R model, object, and ontology.

The top level E-R model is a general model in modeling the reality at an abstract level. It is in accordance with Gruber's best known ontology definition [3] where *an ontology is an explicit specification of a conceptualisation*. An ontology is a specification, namely a pair of $< \Sigma, \Psi >$ to describe that Σ satisfies the axioms Ψ derived from a domain model, where a conceptualisation Σ is a pair of $< \mathcal{C}, \mathcal{R} >$

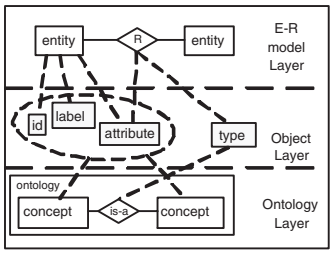

Figure 2. Three-layer ontology model

with $\mathcal{C}$ representing a set of concepts, and $\mathcal{R}$ standing for a set of relations over these concepts. Below the E-R model layer is the object layer where "id", "label", "attribute", and "type" are defined. Objects in this layer are instances of E-R model layer classes. The bottom layer is the ontology layer. Various ontologies of the system are instantiations of objects in the object layer.

A prominent feature of the ontology in this system is the "attribute" of an "entity" or a "slot". It is defined to take both simple and complex data type such as af *class* type.

When an ontology is imported to the system, an *OA* is automatically generated to act on behalf of this ontology and to perform ontology related operations such as ontology access and ontology update. Other agents, such as *functionary agent*s, can not access individual ontologies.

4 Agent Design and Implementation

In this section, the interrelated processes and functionality together with the implementation of each agent in the JADE agent platform[2](http://jade.tilab.com/) are presented. JADE is claimed to comply to FIPA (*http://www.fipa.org*) specifications.

4.1 User Agent

The *UA* assists the user in formulating its requests posting queries (e.g. tasks) to the proposed system via the *IA*, and in visualising the results of required tasks according to the user's requirements. It seperates complex internal system designs and implementations from the user. During initialisation, the *UA* waits for information from the `user interface`. The *UA* must engage in "communications" with the *IA* in fulfilling its role because the internal structure is invisible to it yet. The *UA* only knows the *IA*.

The *UA* knows nothing about other agents created in our multi-agent system environment. The agent interactions only take place between the *UA* and *IA*.

[2]Refer to the website (*http://exp.telecomitalialab.com/*) for more details of why JADE is a suitable technical platform for modelling distributed and heterogeneous environments. Refer to the website (*http://www.agentlink.org/*) for available agent software.

The *UA* waits for actions from the `user interface` and then sends an ACLMessage to the *IA* when it has received a user action. For example, if a user in the mapping interface chooses two agents and then clicks `OK` to execute the `mapping module`, the `user interface` passes the agents to the start mapping method in the *UA*. The *UA* does not know how to map but it creates an ACLMessage and adds the two ontology agents as parameters in the ACLMessage and sends this ACLMessage off to the *IA*. When information is to be displayed to users, the *IA* passes a message back to the *UA* which will show necessary results on the `user interface`.

4.2 Interface Agent

The *IA* acts as an interface between agents in our created MASs and the *UA*. Upon initialisation, every agent knows about the *IA*. So if any other *functionary agent*, for example, the *MA*, wants to show mapping results to the user, it can send a message to the *IA* which will pass it on to the *UA* to display. On the other hand, when new agents, for example, *OA*s, are added in, the *IA* will let all other agents know. So existing agents may refer to newly added agents.

The *IA* facilitates the modularity of the MAS environment. It is involved in all sorts of tasks when communications between created agents (e.g. a multi-agent system) and the `user interface` are required.

The *IA* receives messages from the *UA* and then based on the *conversation id* of the ACLMessage, it passes it on to the appropriate agent in our created multi-agent system to deal with. For example, the *IA* may receive a message from the *UA* with the *conversation id* of "start-mapping". The *IA* does not know anything about mapping but knows when it gets this message to pass it on to the mapping agent to handle. The mapping agent will then extract the content of the message and perform mapping on these two *OA*s.

4.3 Ontology Agent

The *OA* provides an ontology related information and operations to other agents. The *OA*s isolate details of external ontologies (e.g. different ontologies) from *functionary agent*s.

Upon initialisation, it waits for information from other agents. It engages in "communication" with these agents with respect to the required information. This process may loop until the module runs out of concept candidates of a specified ontology. The *OA* may return a result to the user via the *IA* and *UA*.

The *OA* provides as much information of the ontology it acts on as possible by interacting with other agents (Figure 3). The *OA* operates over two structures: (1) the ontology structure (e.g. the ontology it acts on behalf of);

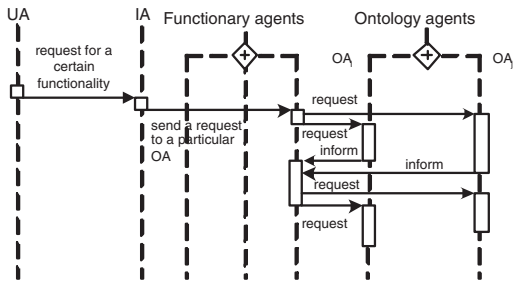

Figure 3. Interactions of `ontology agent` (*OA*) in AUML

and (2) the mapping results (i.e. `mapping.txt`). The major operations on these two structures are as followings.

- operations over the ontology structure
  - *Insert*, namely insert a new concept into the ontology structure, the *UA* may specify where to attach the concept (for ontology integration phase);
  - *Traverse*, namely traverse the structure, return the requested concepts or "True/False" of a given proposition.
- operations over the mapping results
  - *Append*, namely append the mapping results into the mapping result;
  - *Search*, namely search `mapping.txt`.

Besides, *OA*s act properly when triggered by different sources of driving forces. For example, in a query module, an *OA* will dispatch a particular query request to other known *OA*s if it is unable to answer the query. Next, we focus on ontology mapping related interactions, by taking into account the interactions between the *OA* and *MA*, and the *OA* and *QA* as well.

An *OA*'s main responsibility is to look after an ontology, either the one imported from a RDF(s) file or the one created in an object oriented way. *OA*s are all equipped to be able to perform some functions to reify agent behaviours responding to certain coming ACLMessages.

The *OA*s are not added to JADE at start up. They are added to our current running container when the user presses the `run` button in the `user interface`. Assuming that the user wants the system to add two *OA*s, for example, `beer1Agent` and `beer2Agent`. When these agents first enter our system, they register themselves with the *IA*. All agents will have a reference to these two added *OA*s via the *IA*.

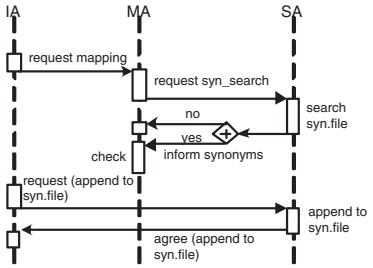

Figure 4. Interactions of `thesaurus similarity agent` (*SA*) in AUML

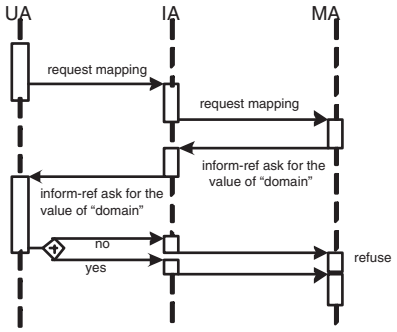

Figure 5. Module of deciding domain in AUML

4.4 Thesaurus Similarity Agent

The *SA* maintains a thesaurus for the purpose of similarity. Upon initialisiation, it waits for information from either a *functionary agent* or the *UA* with respect to query or update the contents in the thesaurus. It then returns the result to the corresponding requesters.

The *SA* may work in two cases by interacting with other agents (Figure 4). One is in the mapping module when the *MA* is looking for synonyms of a given term from one ontology if the *MA* has not found such a concept in another ontology. Another case is when a user asks to update the thesaurus list.

The *SA* holds a list of common words and synonyms of words. For example, during the mapping process, if the *MA* is trying to perform mapping on two ontologies, and gets the concept *suds* from one ontology, and wants to map it to another ontology, but that ontology has no concept called *suds*, then the *MA* sends an ACLMessage to the *IA*. The *IA* reads the `conversation id` of this message and passes it on to the *SA*. The *SA* then looks in its list and gathers a list of synonyms for the particular term. The *SA* then sends this list back in an ACLMessage to the *IA*, which passes it back to the *MA*. The *MA* will then use this list for mapping.

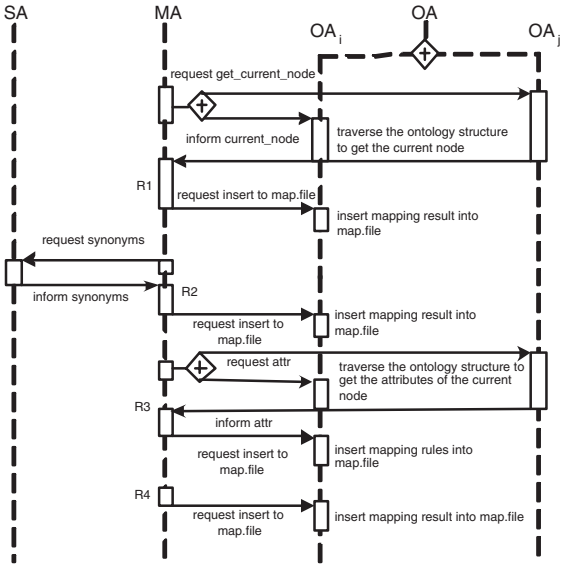

Figure 6. Interactions of `mapping agent` (*MA*) in AUML

4.5 Mapping Agent

Dynamic mapping is thought as on the right track to pave the way for further ontology operations. The *MA* takes effect on receiving a mapping request from the *UA* via the *IA*. It engages in "communication" with *OA*s and the *SA* to execute mapping until the process completes. *OA*s will have a reference to the mapping results.

Unlike the *OA* or other agents, the *MA* does not operate directly over existing structures. It takes effect via the *IA* in the module (Figure 5) in deciding whether existing ontologies come from the same domain or not; or in the module of ontology mapping (Figure 6) through the *OA* and *SA* to acquire relevant information. The former module paves the way for deployment of predefined rules (actually, these rules are the required little prior knowledge)

In reality, the process of deployment of rules may loop until it runs out of sub-concepts of a specified ontology. Mapping is conducted from a particular *OA*'s perspective. In other words, mapping has directions.

The *MA* performs mapping on pairs of ontologies and writes the mapping results out to a mapping file. To illustrate the entire process, assume that the user inputs two ontologies. The `user interface` then sends these two ontology names to the *UA* which wraps them up in an ACLMessage and passes it on to the *IA*. The *IA* will pass the message on (based on `conversation id`) to the appropriate agents to handle.

4.6 Integration Agent

Ontology integration is based on the results of ontology mapping. It is from a particular agent's perspective. In other words, the system has no intention to be engaged in conceptual modelling issues (e.g. how to build an ontology), instead, it uses the existing conceptual models by choosing one (by the user) of them. The purpose of integration is to extend, enlarge, or refine a certain ontology from a global view.

Upon initialisation, the *InA* waits for the action of performing integration over specified ontologies. The *InA* then engages in "communications" with proper *OA*s until the integration task has been solved. Integration module executes integration from a given start point of an ontology. It then requests corresponding *OA*s to traverse sub-concepts of the ontology. Briefly speaking, the *InA* counts the appearance of each concept of existing ontologies, and then filters unexpected concepts with a given threshold. By saying this, we do not mean that we attempt to change the conceptual modelling of the ontology. Instead, ontology integration is based on a specified ontology. The process (shown in Figure 7) looks like:
(1) obtain ontology related information via *OA*s (e.g. access mapping files);
(2) keep the numbers of occurrences of each concept in the specified ontology;

Three cases may take place according to the mapping results. They are:

- Case 1: semantic equivalence for the current two concepts (for example, "beer" is the same as "suds"):

  In this case, increase the number of occurrences of the concept by 1 for each equivalence;

- Case 2: inclusive relation for the current two concepts (for example, "stout" is a kind of "ale"):

  In this case, insert sub-concepts of the counterpart into the specified ontology structure but keep the original relations;

- Case 3: no semantic equivalence for the current two concepts but their corresponding direct ancestors are semantically equivalent:

  In this case, insert the counterpart into the specified ontology but without conflict with existing sub-concepts of the same ancestor;

(3) filter unexpected concepts by a given threshold.

The *InA* performs integration of the ontology agents. To illustrate the entire process, we take the example below. When the user wants to integrate the ontologies they provide the dominating ontology, and a threshold. The `user interface` passes this information on to the *UA* which wraps it up in an ACLMessage and sends it to the *IA*. The *IA* looks at `conversation id` of this message then passes it on to the *InA*. The *InA* will request the ontologies from available *OA*s (this is done again by ACLMessages). When the integration module has all the ontologies (by contacting *OA*s), it performs integration module based on the dominating ontology, the threshold, and of course the mapping results. When the integration has been done, the first thing is that an *OA* is created on the fly which will look after the newly integrated ontology. Then the *InA* sends the new ontology to the *IA* in an ACLMessage. The *IA* then passes it on to the *UA* which will display the new ontology to the user in a tree view. The user can later exports this new ontology to a RDF(s) format.

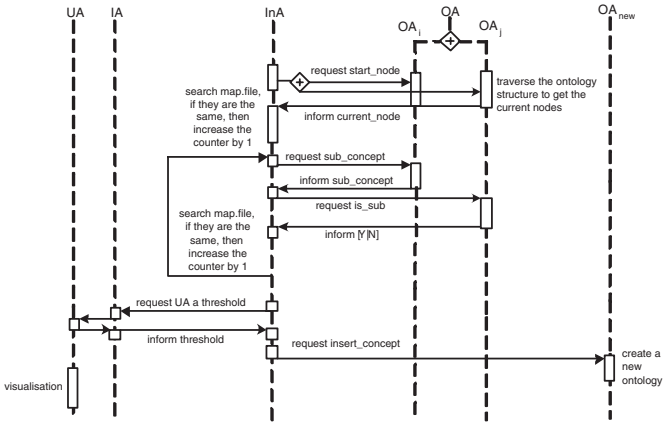

Figure 7. Interactions of `integration agent`(*InA*) in AUML

4.7 Consistency Checking Agent

The *CA* is developed to check the consistency of integrated ontology (assuming all provided ontologies are consistent). Upon initialisation, the *CA*) waits for actions from the `user interface` to perform consistency checking task upon a generated ontology. The *CA* then executes checking module by engaging in "communications" with a particular *OA*. This process may loop until no candidate concept from the ontology left.

When the *CA* initialises, the *CA* waits for a request to perform checking task over a specified ontology. It then engages in "communications" with a particular *OA* to operate the checking module. It returns the user either (a given ontology) consistent or not. If the ontology is inconsistent, the problem part of the ontology is highlighted in the `user interface`. The whole process is shown in Figure 8.

When the `checking` button (to check the consistency

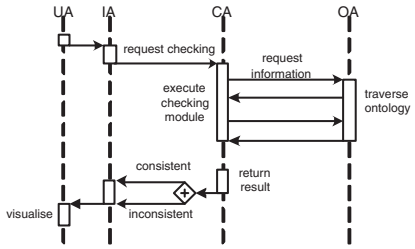

Figure 8. Interactions of `consistency checking agent` (*CA*) in AUML

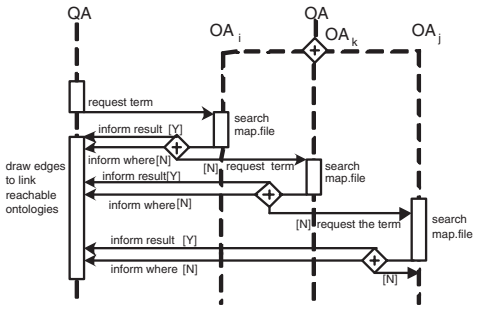

Figure 9. Interactions of `query agent` (*QA*) in AUML

of integrated ontology) of `user interface` is clicked, the `user interface` passes this information on to the *UA*. The message is wrapped up in an ACLMessage and sends to the *IA*. The *CA* gets a ACLMessage from the *IA* to check the consistency of an ontology. It sends a message back stating that inconsistency or no inconsistency is found, which will eventually be passed back to the `user interface`.

4.8 Query Agent

The *QA* is designed to govern query execution around available *OA*s over the mapping results. The query aims to provide an equivalent description of queried term if the agent knows it. If not, the agent then passes the query to other available *OA*s. The query dispatching module continues until either the query is answered or the process has queried all existing *OA*s with respect to the term. The result (e.g. a query routing) is displayed on the `user interface`.

Upon initialisation, the *QA* waits for actions from the `user interface` to perform the query. The *QA* then executes the query module by engaging in "communications" with *OA*s with reference to mapping results. This process completes when the queried terms are interpreted.

The query agent acts like a planning agent for a reachable path in a decentralised ontology network. It is responsible for tasks such as keeping tracks of *OA*s in case of backtracking is required (Figure 9).

The *QA* gets a ACLMessage from the *IA* to query an ontology of a specified term. For example, when the user inputs the query term and the ontology to query, the `user interface` will pass the query term and the ontology to the *UA* which will wrap up this information in an ACLMessage and send it to the *IA*. The *IA* will look at `conversation id` of the message and realise that it needs to send it to an *OA* itself. So the *IA* will unwrap the message to determine which agent it needs to send to. The *IA* then sends the ACLMessage to that *OA*. Each *OA* has been programmed to handle this kind of message. So when an *OA* receives this message, it searches through its own ontology for the query term. If it is unsuccessful, it then searches through appropriate mapping files. It will then pass a result message back to the *IA*, which will pass it on to the *UA*. Then the user will see a message posted on screen either something like "suds = agent1.beer" or "no semantic equivalence".

4.9 Important Notes

All our agents make use of JADE Message Templates in order to perform certain tasks based on certain attributes of the message. For example, an *OA* can send concepts. When the *OA* receives a message, it needs to determine which of these behaviours to execute. The way we do this is by using Message Templates. Message Templates allow us to filter messages based on attributes like `performative type`, `conversation id`, and `sender`, etc.

5 Prototyping

The JADE agent platform is used to build the MAS system. To meet the main tasks of the work, different classes (e.g. concept class, ontology class and agent class) are defined for ontology and agent generation on the fly.

In the prototype, different ontologies, for example RDFs ontologies, are created for demonstration. Correspondingly, different kinds of agents are generated. Among them, there are `user agent` (*UA*), `interface agent` (*IA*), `ontology agent` (*OA*), and *functionary agent*s. In the following, two major operations will be demonstrated in detail.

(1) Ontology Mapping

In the work, mapping attempts to provide mapping results for any further ontology operations, for example ontology integration. Mapping is operated from a source ontology to a target ontology (e.g pairs of ontologies), so the mapping results are stored locally associated with the

source ontology agent. In other words, only this *OA* can access and update the result.

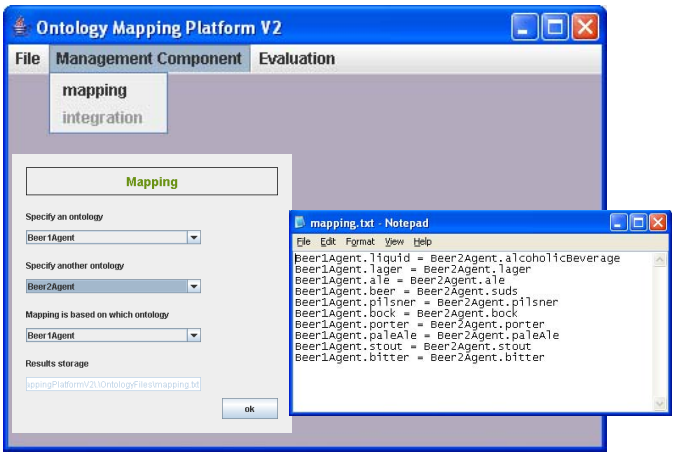

Figure 10. Screen shot of ontology mapping

(2) Ontology Integration

Figure 10 shows the screen shots of ontology mapping. The screen shot at the back is an overall prototype when the "mapping" button is clicked. The lower left screen shot is a mapping screen where the mapping direction is specified in addition to two given ontologies. The mapping result in plain text is shown at the lower right of Figure 10. Ontology integration is based on ontology mapping results in order to provide a global view of existing ontologies in the environment. Since ontology conceptualisation is out of the scope of this work, we will not extend this topic any more but instead to choose one available ontology to perform ontology integration.

Figure 11 shows the screen shots of ontology integration framework and integration results. The upper part is the overall prototype when "integration" button is clicked. The lower left part is the integrated ontology in RDF(s) format, while the lower right part is the integrated ontology in the hierarchical structure.

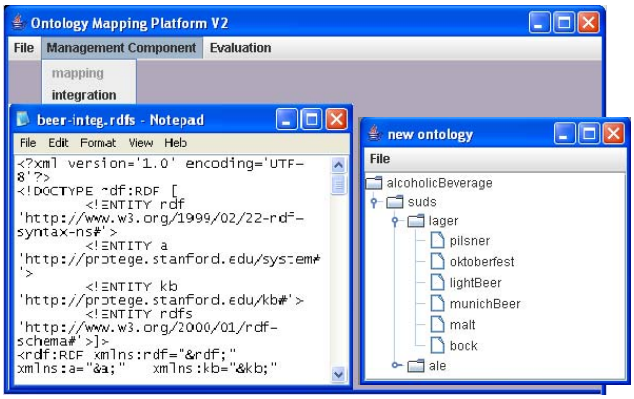

Figure 11. Screen shot of ontology integration

6 Conclusions

In this paper, an agent-based architecture is realised by using the JADE agent platform. Each kind of agent has been detailed and implemented in the proposed framework. We also provide a three-layer user ontology model. We have demonstrated how ontology mapping and integration function in the prototype based on the proposed ontology model.

References

[1] Finin, T., McKay, D., Fritzson, R., and McEntire, R., KQML: An Information and Knowledge Exchange Protocol, *Knowledge Building and Knowledge Sharing*, Kazuhiro Fuchi and Toshio Yokoi (Eds.), Ohmsha and IOS Press, 1994.

[2] Fowler, J., Nodine, M., Perry, B., and Bargmeyer, B., Agent Based Semantic Interoperability in InfoSleuth, *SIGMOD Record*, 28(1), pp. 60–67, 1999.

[3] Gruber, T. R., Toward Principles for the Design of Ontologies Used for Knowledge Sharing, KSL-93-04, Knowledge Systems Laboratory, Stanford University, *http://ksl-web.stanford.edu/*, 1993.

[4] Jennings, N., and Wooldridge, M., Agent-Oriented Software Engineering, in: J. Bradshaw (Eds.), *Handbook of Agent Technology*, AAAI/MIT Press, 2001.

[5] Li, L., Wu, B., and Yang, Y., Semantic Mapping with Multi-Agent Systems, *Proc. of the IEEE International Conference on e-Technology, e-Commerce and e-Service* (EEE'05), pp. 54-57, Hong Kong, March 2005.

[6] Li, L., Agent-based Ontology Management Towards Interoperability, PhD thesis, Swinburne University of Technology, Australia, 2006. *http://www.it.swin.edu.au/personal/lli/thesis.pdf*.

[7] Noy, F. N., and Musen, M. A., The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping, *International Journal of Human-Computer Studies*, 59(6), pp. 983-1024, 2003.

[8] Odell, J., Parunak, H. V. D., Bauer, B., Extending UML for Agents, *Proc. of the Agent Oriented Information Systems Workshop (AOIS) at the 17th National Conference on Artificial Intelligence*, in G. Wagner, Y. Lesperance, and E. Yu, editors, Austin, Texas, pp. 3-17, 2000.

[9] Wooldridge, M., *An Introduction to MultiAgent Systems*, John Wiley & Sons, 2002.