

Article

Agent-based ontology mapping and integration towards interoperability

Li Li^{1,2} and Yun Yang²

(1) CSIRO ICT Centre, GPO Box 664, Canberra, ACT 2601, Australia

E-mail: Lily.li@csiro.au

(2) Faculty of Information and Communication Technologies, Swinburne University of Technology, PO Box 218, Hawthorn, Melbourne, VIC 3122, Australia

E-mail: yyang@groupwise.swin.edu.au

Abstract: *Interoperability is an important issue in ontology research. In this paper, a novel agent-based framework for managing ontologies in a dynamic environment is developed. The framework has several key characteristics such as flexibility and extensibility that differentiate this research from others. Based on the proposed framework, ontology mapping and integration are investigated. It is believed that inter-ontology processes like ontology mapping with logical semantics are foundations of ontology-based applications. Accordingly, several types of semantic relations are proposed and corresponding mapping mechanisms are developed. Based on mapping results, ontology integration is developed to provide abstract views for participating organizations in the presence of a variety of ontologies. A prototype is built to demonstrate the design and functionalities and is applied to beer ontologies. The prototype shows that the framework is not only flexible but also practical. All agents derived from the framework exhibit their behaviours as expected.*

Keywords: ontology mapping, ontology integration, ontology interoperability, multi-agent systems

1. Introduction

Ontology mapping and integration towards interoperability are important for managing ontologies based on certain applications. An ontology is an *explicit specification of a conceptualization* (Gruber, 1993a). Ontologies are widely used as data representations for knowledge bases and marking up data on the emerging semantic web (Berners-Lee *et al.*, 2001). In reality, the large amount of data which exists in many disparate sources requires unification based on ontologies to allow data integration. Therefore, intelligent techniques for managing ontologies are essential for any practical and general solutions for knowledge-based systems. The vision of ontology for different purposes, such as web-enabled applications, web seman-

tics and information systems, is receiving increasing attention from both academia and industry. In the context of knowledge sharing, an ontology is a specification used for making an ontological commitment which is an agreement to use a vocabulary in a way that is consistent with respect to the theory specified by an ontology (Gruber, 1993a, 1993b). Ultimately, ontologies enable data/information integration in the context of semantics. They would widen the accessibility of information by allowing the use of multiple ontologies belonging to diverse organizations. The benefits of using ontologies have been recognized in many areas such as knowledge and content management, e-commerce and the semantic web. Approaches based on ontologies have shown the advantages for both information integration

and system interoperability including reusability, extensibility, verification analysis and so on. In this paper, we present a novel agent-based approach for the purpose of managing ontologies dynamically in a distributed and heterogeneous environment.

Based on past experiences, e.g. CYC (Lenat & Guha, 1990), a monolithic ontology is unlikely to be constructed. This is due to its large scale, privacy needs, dynamics and heterogeneity. As a typical example, the proliferation of Internet technology and globalization of business environments give rise to the advent of dynamic virtual alliances (Byrne *et al.*, 1993) such as virtual organizations (VOs) among complementary organizations. It is hardly conceivable that a single ontology can be applied to all involved parties and applications. In addition, in numerous domains there is always a need to deal with *different purposes* to keep their uniqueness. However, it seems that no satisfactory solutions to semantic mapping exist because today the vast majority of semantic mappings are still created manually. The slow and expensive manual creation of mappings has now become a serious problem in building ontology-based applications. The problem becomes even more critical when a changeable environment (e.g. the web) is involved, in which ontologies and ontology-based applications might proliferate and scale up. Also a changeable environment enforces underlying ontologies to evolve over time. Finally, the development of technologies such as the semantic web will further fuel deployment of ontologies to enable knowledge sharing and reuse across different sources. Manual mapping is simply not possible for such scales.

When heterogeneity, distribution, autonomy and evolution are concerned with ontologies, achieving effective manual interoperability seems unlikely. Agents interact when there is a demand so that they can understand each other. This provides a potential solution to addressing environment change, which is treated with a flexible approach and seen as a key technology at the knowledge view level. As understanding in a particular domain is required before any

decision is made, agent interaction is regarded as a feasible and effective way to find certain semantics in a certain business scenario at runtime. Besides VOs, the very viable and rapid growth of the web has made it even more prominent than before. Addressing ontology mapping and integration in a timely fashion by providing a conformance view for participating organizations at an abstract level is thus seen as critical. Therefore, runtime interactions are needed for ontology mapping and integration. However, existing knowledge management cannot cope with these interactions effectively as they may occur at unpredictable times between different parts of ontologies.

Given that semantic mapping is a fundamental step in numerous ontology-based applications, providing a global view of existing ontologies in such an environment at an abstract level is another challenge. It is time to think about the development of solutions (e.g. *ontology mapping* and *ontology integration*) which are flexible, robust and applicable to meet the requirements of dynamic changes in an environment.

Multi-agent systems (MASs) (Wooldridge, 2002; Luck *et al.*, 2004) fit well in the situations where flexibility, robustness and applicability are needed. An agent-based perspective is thus an appropriate approach for ontology mapping and integration. This can be applied where operations on diverse ontologies cannot be achieved without considering dynamics and distributions. In this paper, innovative agent-based ontology mapping and integration are developed to provide new approaches to cope with dynamic ontology change in an environment, especially in the context of the web.

The rest of the paper is organized as follows. Section 2 introduces problems in current ontology mapping and integration. Section 3 presents ontology mapping and integration related definitions. Section 4 discusses a novel agent-based framework for ontology management. Section 5 investigates the ontology mapping process and mechanisms under the proposed framework. A prototype is demonstrated and an analysis is given to evaluate the agent-based mapping

mechanisms. Section 6 explores the ontology integration process and mechanisms based on mapping results. An important aspect of integration, namely consistency checking, is discussed and a prototype is presented. Section 7 is a discussion, and Section 8 overviews related work. Finally, Section 9 concludes the paper and points out future work.

2. Problems in current ontology mapping and integration

Reviews on the state of the art of ontology mapping and integration clearly show that existing methodologies and the corresponding systems or tools are lacking in flexibility and extensibility. These methodologies and systems mainly treat ontology hosting environments statically rather than in a dynamical and distributed fashion. Therefore this leads to those systems operating under limited conditions. Some of the causes of the problems are as follows.

- (1) Most architectures used to develop these systems have failed to anticipate future requirements of ontology management. In fact, they are 'functional' systems instead of 'reactive' ones that exhibit intelligent behaviours. Unfortunately, the desired architectures for handling ontology management are not 'functional' enough to handle the required complicated interactions. They are also unable to communicate with high-level protocols and languages.
- (2) Significant prior knowledge is required to enable these systems to operate correctly in ontology management. Knowledge provided in advance does not always hold in a changing environment. This means that these systems are brittle in the sense that human intervention is required to compensate for even slight shifts in the environment. Obviously, they are unable to cope with the requirements of ontology management in an open environment.
- (3) Ontologies in these systems or tools are the only objects. In other words, these systems or tools manage ontologies from the

knowledge management perspective rather than from the knowledge deployment perspective. However, it is known that ontologies are used to provide interoperability among ontology-based applications and are eventually used for reasoning purposes.

In short, existing systems or tools are lacking the capabilities to deal with highly changing environments with their underlying developing motivations. They have neither theoretical backgrounds, clear architectures nor flexibility. Agents in an MAS are autonomous and flexible enough to interact in order to achieve defined objectives (Wooldridge, 1997).

An MAS is a collection of autonomous agents working together to solve problems that are beyond the overall capability of individual agents. The advantages of deploying an agent-based approach in ontology management are threefold:

- (1) the ability to support rational changes in the environment, to have the capabilities to perceive any changes, and to adapt to them accordingly;
- (2) the ability to exhibit goal-directed behaviour to satisfy their design objectives; and
- (3) the ability to exhibit intelligent behaviour by interacting with other agents to achieve designed goals in the environment.

For this reason agents are being advocated as a next generation model for engineering complex and distributed systems (Wooldridge, 1997; Jennings & Wooldridge, 2001).

With the above objectives achieved, problems such as lack of flexibility and extensibility are expected to be resolved.

3. Definitions

We follow Gruber's (1993a) well-known ontology definition in this paper. Ontologies are enabling technology for the semantic web (Berners-Lee *et al.*, 2001). Within the ontology definition, we define an ontology with a specific domain model, \mathcal{T} . Thus a conceptualization Σ is a triple of $\langle \mathcal{C}, \mathcal{R}, \Psi \rangle$, where \mathcal{C} represents a set of concepts, \mathcal{R} stands for a set of relations

and Ψ represents a set of axioms that constrain the meaning of concepts and relationships. Let $\mathcal{C} = \{c_1, c_2, \dots, c_n\}$ be concept denotation names, and let $\mathcal{R} = \{r_1, r_2, \dots, r_m\}$ be some relationships over those concepts. A specification is a pair of $\langle \Sigma, \mathcal{I} \rangle$ such that different instances of the concept exist because an ontology is not independent of the representation and language used to describe it. There are many representations and languages¹ available for encoding an ontology. However, in order to establish the notation used by ontology mapping and integration in this paper, more precise descriptions are given below.

Definition 1 (Ontology morphism) Given two ontologies, a morphism between these ontologies describes a way to associate one ontology to another. A set of morphisms is denoted M^o .

Definition 2 (Ontology) An ontology $\mathcal{F}(\Sigma)$ is defined as

- (1) including an initial rough ontology;
- (2) enabling more complicated ontologies to be formed repeatedly by using available ontologies and their morphisms.

Given $\mathcal{A}_i, \mathcal{A}_j (i, j \in \mathbb{N}, \mathbb{N}$ natural numbers) any subsets of an ontology, $op \in M^o$, an ontology can be precisely defined as

- (1) $\mathcal{F}(\Sigma)_0 = \mathcal{F}(\Sigma_0)$;
- (2) $\mathcal{F}(\Sigma)_{n+1} = \mathcal{F}(\Sigma)_n \cup \{op(\mathcal{A}_i, \mathcal{A}_j) | \mathcal{A}_i, \mathcal{A}_j \in \mathcal{F}(\Sigma)_n\}$

Therefore, an ontology $\mathcal{F}(\Sigma)$ can be represented by

$$\mathcal{F}(\Sigma) = \bigcup_{n \geq 0} \mathcal{F}(\Sigma)_n$$

¹There are many formal languages, such as RDF(s) (<http://www.w3.org/RDF/>), OIL (<http://www.ontoknowledge.org/oil/>), DAML + OIL (<http://www.w3.org/TR/daml+oil-reference>) and OWL (<http://www.w3.org/TR/owl-guide/>), for ontology representation. Intuitively, these languages differ in their terminologies and expressiveness. The ontologies that they model essentially share the same features.

Obviously, there exists

$$\mathcal{F}(\Sigma)_0 \subseteq \mathcal{F}(\Sigma)_1 \subseteq \dots \subseteq \mathcal{F}(\Sigma)_n \subseteq \mathcal{F}(\Sigma)_{n+1} \\ \subseteq \dots \subseteq \mathcal{F}(\Sigma)$$

Assume that $c_i \in \mathcal{C}_i$ and $c_j \in \mathcal{C}_j$ are two different concepts; the following definitions are related to ontology mapping and integration. Note that all these are dealt with at the conceptual level.

Definition 3 (Equivalent) Two concepts are semantically equivalent if $\exists c_i, c_j$ such that $c_i \sim c_j$. Namely, these two concepts

- have the same denotation names (e.g. labels);
- are synonyms; or
- have the same attributes.

Definition 4 (Inclusive) Two concepts are semantically inclusive if $\exists c_i, c_j (c_i \in \mathcal{C}_i, c_j \in \mathcal{C}_j, i, j \in \mathbb{N})$ such that $c_i \leq c_j$ (e.g. c_i is a kind of c_j) or $c_i \geq c_j$ (e.g. c_j is a kind of c_i). Namely, the attributes of one concept are also the attributes of the other. In this paper, c_i is also called the sub-node of c_j if $c_i \leq c_j$. Alternatively c_j is called the super-node of c_i . For simplicity, it is denoted either $c_i = sub(c_j)$ or $c_j = super(c_i)$.

Definition 5 (Ontology mapping) A mapping \mathfrak{R} between two ontologies $\mathcal{F}_i(\Sigma_i)$ and $\mathcal{F}_j(\Sigma_j)$ exists if $\exists c_i, c_j$ such that $\mathfrak{R}(c_i, c_j) \in \{\sim, \leq, \geq\}$. In some cases, we can only achieve partial mapping (i.e. mappings exist between some pairs of concepts in two ontologies) instead of full mapping (i.e. mappings exist between all pairs of concepts and relations in two ontologies).

Definition 6 (Concept family) Concept family is denoted as Σ^* (over Σ). Given a set of concepts \mathcal{C} , there exists $\mathcal{C} \subseteq \Sigma^*$.

Definition 7 (Concept kernel) For any subset \mathcal{C}' of \mathcal{C} , obviously it satisfies $\mathcal{C}' \subseteq \mathcal{C} \subseteq \Sigma^*$. The concept kernel \mathcal{C}' varies from time to time in applications.

Attempting to build an integrated ontology which serves at a higher application level than that of various others in ontology repositories is challenging. This is because of the difficulty in reusing available source ontologies effectively whenever semantic integration is concerned. In this paper, ontology integration consists of a closure of the existing ontologies that satisfies the above properties (1) and (2) of Definition 2, and the rest of the specific ontology whenever needed.

Definition 8 (Ontology integration) Normally there is a particular domain model \mathcal{T}_i and a family of concepts Σ_i^* with \mathcal{C}_i as a concept kernel. This particular ontology is denoted as $\mathcal{T}_i(\Sigma_i^*)$. With existing ontologies denoted as $\mathcal{T}_1(\Sigma_1), \mathcal{T}_2(\Sigma_2), \dots, \mathcal{T}_k(\Sigma_k)$, \mathcal{T}_i closure, denoted as \mathcal{T}^ξ , is defined by

$$\mathcal{T}^\xi = \cap \{ \mathcal{B}_j | \mathcal{C}_i \subseteq \mathcal{B}_j \wedge (\mathcal{T}_i(\mathcal{B}_j) \text{ holds}), \\ \mathcal{B}_j \subseteq \mathcal{C}_i, j \in [1, \dots, k] \}$$

as long as $\mathcal{T}_i(\Sigma_i^*)$ holds.

Eventually, the integrated ontology $\mathcal{O}_\mathcal{J}$ can be formalized by

$$\mathcal{O}_\mathcal{J} = \mathcal{T}^\xi \cup \mathcal{C}_i \setminus \mathcal{C}_i'$$

For the purpose of ontology integration, we need to consider the consistency issue. It is obvious that the above definitions allow us to check the consistency of a newly derived ontology. The ontology disjoint and ontology consistency are defined as follows.

Definition 9 (Disjoint) Two concepts are disjoint if $\exists c_i, c_j$ such that $c_i \cap c_j = \emptyset$.

Definition 10 (Consistent) Given $\forall c_i, c_j (c_i \neq c_j)$, ontology consistency is defined by

- (1) $c_i \cap c_j \neq \emptyset$;
- (2) $\forall c'_i \geq c_i, c'_j \geq c_j$, then $\bigcup_{c'_i \in C} super(c_i) \cap \bigcup_{c'_j \in C} super(c_j) \neq \emptyset$.

Definition 11 (Ontology interoperability) This is a full realization of a vision in which at least

two conditions need to be met: (1) any two ontologies $\mathcal{T}_i(\Sigma_i)$ and $\mathcal{T}_j(\Sigma_j)$ are interoperable, i.e. $\mathcal{T}_i(\Sigma_i) \longleftrightarrow \mathcal{T}_j(\Sigma_j)$, if $\exists \mathcal{R}$ (direct mapping) between these two ontologies, or $\exists \mathcal{R}', \mathcal{R}'', \dots$ (indirect mappings) which lead to the mapping of these two ontologies; and (2) the ontologies comply with ontology mapping and integration as defined above.

4. Novel agent-based framework for ontology management

So far, we have argued that an agent-based approach is more suitable because it is what ontology management needs. In this section, a general agent-based framework is proposed, followed by a brief introduction to the behaviours of each kind of agent and their communication.

A framework for ontology management must be flexible to allow agents to go online or offline freely. The flexibility of the system becomes evident when applications become more and more complex. An MAS is well suited to model ontology management on the web. The overall framework of agent-based ontology management is shown in Figure 1. The behaviour of each kind of agent in this environment is briefly described below. These will be discussed in more detail in the succeeding sections.

- **User agent (UA):** A user agent is designed to know only the interface agent (IA). This agent interacts with a particular GUI. This includes getting the business scenario from the GUI and passing it on to IA to display the proper results on the user interface (e.g. GUI) when it receives a return message from IA.
- **Interface agent (IA):** This agent interacts with the UA, which includes getting the business scenario from a particular GUI and passing it on to the virtual community,² and then presenting the UA with the expected results. It acts as a broker in an

²A virtual community is a group of partners commonly interested in a certain business scenario.

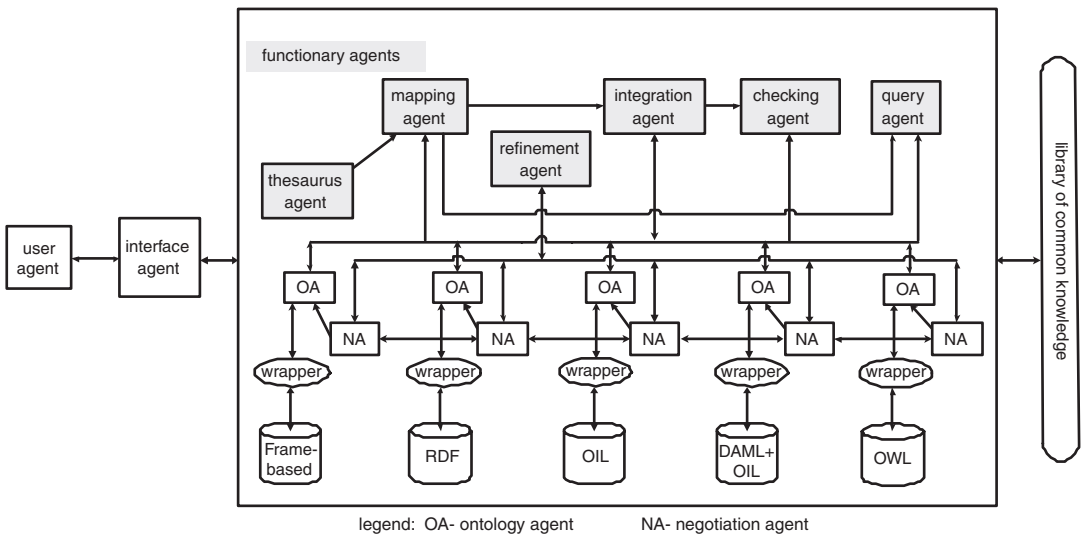


Figure 1: *Agent-based framework.*

attempt to help various agents find each other in a distributed environment.

- **Ontology agent (OA):** This agent acts on behalf of a certain ontology. It behaves properly in a specified agent platform. It is equipped with the functionalities of a certain ontology, e.g. it operates over the ontology structure and a particular intermediate result structure. The main purpose of an OA is to perform ontology-related tasks which are isolated from external *functionary agents*. The presence of an OA allows flexible system organization.
- **Negotiation agent (NA):** This agent takes part in negotiation setting with an attempt to obtain evolving ontology information for the corresponding OA.
- **Functionary agents (FAs):** This is a group of agents which provide the functionalities of thesaurus similarity measure, ontology mapping, ontology integration, ontology refinement, consistency checking and querying. These roles are described below.
- **Thesaurus (similarity) agent (SA):** This is responsible for maintaining thesaurus similarity within a suitable structure. The major tasks of an SA are to (1)

append new concepts to the structure, (2) search for 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 management towards interoperability. The major task of an MA is to estimate whether two given concepts map to each other according to its knowledge.
- **Refinement agent (RA):** This maintains ontology coherence and integrity in an environment where they may change frequently. The major tasks of an 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.
- **Integration agent (InA):** This 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 unexpected items before sending them to the OA.

- (Consistency) **Checking agent (CA)**: This checks the consistency of an ontology. The major tasks of a CA are to (1) check if there exists a particular concept which is a subclass of two disjointed concepts and (2) return Boolean values accordingly.
- **Query agent (QA)**: This shows how to utilize mapping results with distributed ontologies. The main tasks of a QA are to (1) respond to any concept-like queries, (2) dispatch the query to other recognized agents in the environment if the RA is unable to do it by itself and (3) keep the query path.
- **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 during runtime through agent communications.
- **Ontology representations (wrapper)**: Due to the diversity and heterogeneity of ontologies, it is not unusual that different kinds of representation languages coexist, e.g. RDF, OWL. Each ontology representation has a corresponding wrapper to translate a particular ontology representation into a common language.

In this framework, the fact that different agents work collaboratively is highlighted. For example, FAs can effectively access ontologies via OAs, whilst an MA (one of the FAs) may contact an SA (another of the FAs) or any other agents which can provide required functionalities. A scenario can be as follows. A user queries the system through the UA. Then the UA converts the user's query to a suitable representation. After that the IA collects relevant information from the previous process and passes it on to the corresponding FAs. After a series of operations, a final result is provided graphically to the user via the UA. Without any doubt, ontologies are an excellent foundation for agent communication.

In our proposed agent-based framework, there are five major kinds of agents. They are

(1) OAs; (2) NAs; (3) FAs consisting of an SA, an MA, an InA, an RA, a CA and a QA; (4) an IA; and (5) a UA. As stated in Wooldridge (1997) and Jennings (2000) agents in MASs are autonomous and can engage in flexible and highly interactive actions. In fact, agents are able to perceive their environment and respond in a timely fashion to changes that occur. They are also capable of interacting with other agents (and possibly humans) in the goal of achieving their designated objectives. In other words, agents are highly interactive in order to satisfy their objectives. In detail, their abilities of reactivity, proactivity and sociability ensure that they can work together properly and rationally to achieve their goals beyond their individual capabilities and knowledge.

An agent in this research is defined as having amongst its mental attitudes a goal G and an intention I . Suppose that in some cases the agent itself cannot carry out the intention. The question then becomes how to work with others, whom to send messages to and which messages should be sent. According to its intention of satisfying a goal, it sends out messages to those agents to result in an expected outcome or a rational effect on its goal (see Figure 2).

In Figure 2, $agent_i$ sends out a message to $agent_j$ in order to achieve its designed objective. This can be demonstrated by the UA sending a message to the IA when it receives a user request. Assuming that the user is using the integration interface and chooses some agents to execute integration, the UA passes the selected agents to the *start-integration* method in the UA. Because the UA has no knowledge of the integration process, it creates a message, adds specified OAs as parameters to the message and sends it to the IA. The IA then passes the message on to a particular FA which claims that it has the required capabilities for tackling the corresponding task.

Agent communication is based on the FIPA (<http://www.fipa.org>) Agent Communication Languages (ACLs). ACLs provide agents with a means of exchanging information and knowledge, which is the essence of all forms of interaction in MASs. Due to message overheads and

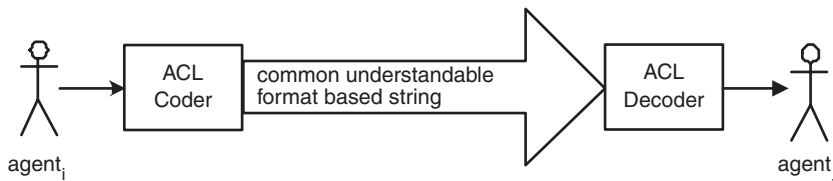


Figure 2: *Agent communication channel.*

the possibility of network congestion, normally the message is parsed into a common understandable format based string. The presence of an ACL Coder and an ACL Decoder at each end of the communication is to convert the message to a *String* and vice versa.

At an abstract level, agents work together based on interactions (see Figure 2). Let us assume that individual agents cannot carry out the intention by themselves. If agents behave rationally, they will contact other agents which will satisfy the intention and thus achieve the end goal. An interaction process (e.g. Figures 3, 4 etc.) is needed as interactions between agents may take place concurrently. After investigating the existing tools, we have chosen AUML (<http://www.auml.org/>) to describe the artifacts of agent interactions.

5. Ontology mapping

Ontology mapping is the baseline to achieve ontology interoperability. This section addresses the mapping process and mapping mechanisms under the proposed framework. This is followed by a prototyping and evaluation discussion.

5.1. Mapping process

In our proposed framework (Li, 2006), there are different agents to represent the perspectives of competing but coordinating organizations, and the distributed nature of the problems (multiple organizations with various ontologies). Suppose an OA, which acts on behalf of the corresponding ontology, is responsible for ontology-related tasks. It provides as much information about the ontology it acts on as possible. The OA

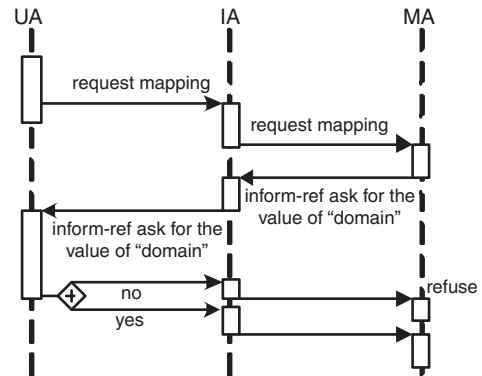


Figure 3: *Process for deciding the domain in AUML.*

operates over two structures: (1) the ontology structure (e.g. an ontology this agent acts on behalf of) and (2) the mapping results.

Unlike the OA or other agents, the MA does not operate directly over existing structures. Instead it operates via the IA as defined in AUML (Bauer *et al.*, 2001). This is done in the process³ (Figure 3) of deciding whether existing ontologies come from the same domain or not; or in the process of ontology mapping (Figure 4) through the OA and SA to acquire relevant information. The former paves the way for the deployment of predefined rules in the latter. Actually, these rules require little prior knowledge (see Definitions 3 and 4 in Section 3). The rules are as follows.

R₁: If two concepts have the same labels, they have the same meanings, i.e. they are semantically equivalent.

³'inform-ref', 'request' in Figure 3 and 'inform' in Figures 4 and 7 are performatives of FIPA ACLs.

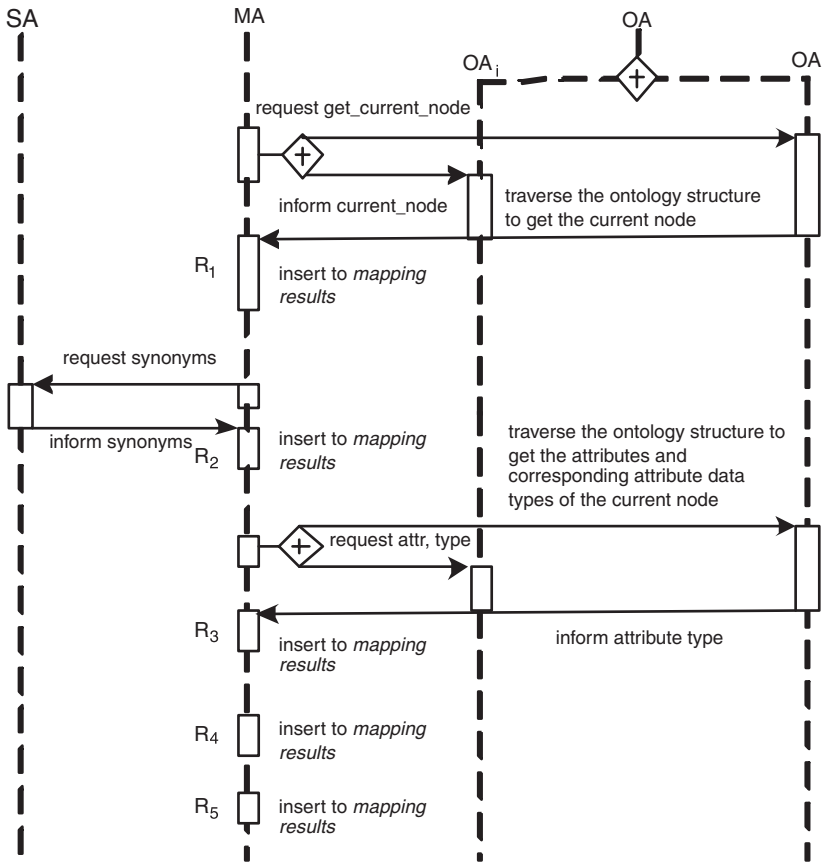


Figure 4: Interactions from MA's view in AUML.

- R₂: If the labels of two concepts are synonyms, they are semantically equivalent.
- R₃: If two concepts have the same attributes as each attribute of the same *datatype*, i.e. if pairs of $\langle \text{attribute}, \text{datatype} \rangle$ are the same, they are semantically equivalent.
- R₄: If all pairs of $\langle \text{attribute}, \text{datatype} \rangle$ of one concept are also pairs of another concept, there is a class-subclass relationship between them.
- R₅: If super-concepts are the same (according to R₁, R₂ and R₃), sub-concepts of one node (e.g. one super-concept) are also sub-concepts of another node (e.g. another super-concept), without conflicting with each other.

In Figure 4, a scenario starts with the MA sending a particular request to the existing OAs to obtain the concepts it requires. The MA then performs the mapping algorithm to check whether the given concepts are semantically equivalent or inclusive. Now, let us take a closer look at the mapping process presented in Figure 4.

An MA is responsible for the ontology mapping of related tasks while working with OAs. The mapping algorithm starts with the MA sending requests to the OAs (for the current concept) for the given ontologies (for simplicity, interactions between the UA and IA, IA and MA are omitted). It then checks if two acquired concepts satisfy the rules in its knowledge base. The similarity

measurement is defined as

$$sim(c_1, c_2) = \sum_{j=1}^k \frac{w_j sim_j(c_1, c_2)}{k'}$$

where $j, k, k' \in \mathbb{N}$, $k' \leq k$, k' represents the number of non-zero similarity measurements of a specific method, w_j ($w_j \in [0, 1]$) is the weight for a specific method, and $sim_j(c_1, c_2)$ is the similarity measurement of a specific method. When the summarizing similarity of two concepts from similar ontologies is greater than a given threshold δ , the MA sends a request to a specific OA to insert the mapping results in a data repository. The process may loop and wait to be enacted for further acquired concepts 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. This is outlined as follows:

- (1) obtain ontology-related information via OAs;
- (2) estimate obtained information according to the knowledge in the MA's knowledge base, i.e. the MA computes whether semantic relations between pairs of ontologies exist and if so what kind;
- (3) write to the mapping results.

Following the above approach, mapping results are available for further ontology operations such as ontology integration discussed in Section 6.

5.2. Mapping mechanisms

The mapping process operates over available ontologies in order to achieve ontology interoperability. The MA is in charge of mapping tasks in the environment. The process uses the following functions or data structures to check whether two given concepts meet the predefined rules.

- **initialize**: Initialize the process.
- **next-c**: Request a particular OA for the next concept of a specified ontology. It returns the concept if it exists.

- **next-a**: Request a particular OA for the attribute of a specified concept. It returns the attribute if it exists.
- **next-t**: Request a particular OA for the corresponding datatype of a certain attribute of a specified concept. It returns the datatype if it exists.
- **comp**: Compare two items. It returns *true* if two compared items are equal. Otherwise, it returns *false*.
- **syn**: Request the SA for the synonyms of a given concept. It returns a synonym list (e.g. *syn-list*) if it exists.
- **measure-sim**: Compute the weighted similarity over the given methods (see Section 5.1 for the calculation formula).
- **add**: Add items to the mapping results.
- **next-i**: Request an equivalent relation from the mapping results.
- **add-sub-concept**: Add inclusive relations to the mapping results.

The pseudocode of the mapping algorithm is shown below. This algorithm may execute repeatedly until existing ontologies have been mapped whenever needed.

Pseudocode of mapping algorithm

Assume the mapping algorithm starts from a given start point.

O_1, O_2 : source ontology and target ontology, respectively (for simplicity, an ontology in the algorithm is denoted as O_j instead of $\mathcal{T}(\Sigma)$)

c_1, c_2 : concepts from these given ontologies, respectively

att_1, att_2 : attributes of c_1, c_2 , respectively

$type_1, type_2$: datatypes of att_1, att_2 , respectively

$flag_1, flag_2$: flags with Boolean values

syn-list: a list of synonyms of a specified concept label returned from function *syn*

w_1, w_2, w_3 : weights for specific methods

sim_1, sim_2, sim_3, sim : similarities

mapping-results: a data repository to record mapping results

δ : given threshold to filter out inappropriate mappings

```

1.  Function mapping {
2.  initialise;
3.  do {
4.       $c_1 = \text{next-c}(O_1)$ ;
5.       $c_1 = \text{next-c}(O_1)$ ;
6.      if  $!(\text{comp}(c_1.\text{label}, c_2.\text{label}))$  {
7.           $\text{syn}(c_1.\text{label})$ ;
8.      } else {
9.           $\text{sim}_1 = \text{measure-sim}(R_1)$ ;
10.     } //end if
11.      $\text{flag}_1 = \text{flag}_2 = \text{true}$ ;
12.     while  $(\text{syn-list}(c_1.\text{label}) \neq \text{Null})$  {
13.         if  $!(\text{comp}(\text{syn-list}(c_1.\text{label}), c_1.\text{label}))$  {
14.              $\text{flag}_1 = \text{false}$ ;
15.         } else {
16.              $\text{sim}_2 = \text{measure-sim}(R_2)$ ;
17.         } //end if
18.         if  $!(\text{flag}_1)$  {
19.             while  $(\text{next-a}(c_1) \neq \text{Null} \ \&\& \ \text{next-a}(c_2) \neq \text{Null})$  {
20.                  $\text{att}_1 = \text{next-a}(c_1)$ ;
21.                  $\text{att}_2 = \text{next-a}(c_2)$ ;
22.                  $\text{type}_1 = \text{next-t}(c_1)$ ;
23.                  $\text{type}_2 = \text{next-t}(c_1)$ ;
24.                 if  $!(\text{comp}(\text{att}_1, \text{att}_2) \ \&\& \ \text{comp}(\text{type}_1, \text{type}_2))$  {
25.                      $\text{flag}_2 = \text{false}$ ;
26.                 } else {
27.                      $\text{sim}_3 = \text{measure-sim}(R_3)$ ;
28.                 } // end if
29.             } //end while
30.         } //end if ( $\text{flag}_1$ )
31.         if  $(\text{flag}_2)$  {
32.             if  $(\text{next-a}(c_1) \neq \text{Null} \ \&\& \ (\text{next-a}(c_2) = \text{Null}))$  {
33.                  $\text{add}(\text{mapping-results}, (c_1, c_2, \sqsupset))$ ;
34.             } // end if
35.             if  $(\text{next-a}(c_1) = \text{Null} \ \&\& \ (\text{next-a}(c_2) \neq \text{Null}))$  {
36.                  $\text{add}(\text{mapping-results}, (c_1, c_2, \sqsubseteq))$ ;
37.             } // end if
38.         } //end if ( $\text{flag}_2$ )
39.          $\text{sim} = \sum_{j=1}^3 \text{measure-sim}(R_j)$ ;
40.         if  $\text{sim} \geq \delta$  {
41.              $\text{add}(\text{mapping-results}, (c_1, c_2, =))$ ;
42.         } // end if
43.     } // end while
44. } while  $(\text{next-c}(O_1) \neq \text{Null} \ \&\& \ \text{next-c}(O_2) \neq \text{Null})$ ; //end do
45.     while  $(\text{next-i} \neq \text{Null})$  {
46.         add-sub-concept;
47.     } //end while
48. } // end function

```

Lines 3–10 relate to R_1 , lines 11–17 to R_2 , lines 18–30 to R_3 , lines 31–38 to R_4 and lines 39–42 are for an overall similarity measurement leading to equivalent relations. At the end, lines 45–47 relate to R_5 .

5.3. Prototype

Jade⁴ is used as a technical platform to implement ontology mapping. The running example ontologies come from the domain of beer and concern its different types. The first one is from the DAML ontology library (<http://www.daml.org/ontologies/66>). A fragment of this is shown in Figure 5(a). The second is built on the definition of the term ‘beer’ from WordNet (<http://wordnet.princeton.edu/>). A fragment of it is shown in Figure 5(b). The third is based on basic types of beer provided by the website http://www.dma.be/p/bier/1_2_uk.htm#. A fragment of this classification is shown in Figure 5(c). The fourth is an Australian beer types ontology based on information from the websites http://www.australianbeers.com/beers/beer_types/beer_types.htm and http://www.fosters.com.au/beer/about/beertypes/beer_types.asp. A fragment of this classification is shown in Figure 5(d). Our main interest is in the term beer and the corresponding hyponym relationship. These four ontologies are about types of beer, but from different points of view.

In the running example ontologies, we assume that \mathcal{R} consists of relations $\{\textit{hyponym}(\textit{is-a}), \textit{part-of}\}$. Moreover, we assume that participating ontologies agree upon a small set of commonsense knowledge regardless of syntactical, structural and language heterogeneity to conform to a normalized uniform representation. Generally speaking, the taxonomy is at the heart of ontologies and ontology applications. So we also assume that ontologies have the same structures (e.g. hierarchical). Moreover, we suppose that there are no polysemous words in an ontology, and there is only one possible relation between two concepts of the ontology. Finally,

⁴Jade (<http://jade.tilab.com/>) is a technical platform for modelling distributed and heterogeneous environments. Jade is claimed to comply to FIPA specifications.

we assume that different ontologies are available and there is no need to consider conceptual modelling issues ourselves.

These four examples are used to demonstrate the mapping mechanisms in an MAS environment. Different kinds of agents (e.g. a UA, an IA, an MA, a QA and an OA) have been created to take different roles in ontology mapping.

Ontology mapping is executed between pairs of ontologies. For example, mapping is done between two specified ontologies O_1 and O_2 . Since there is always an agent to act on behalf of the corresponding ontology in the algorithm, O_1 and OA_1 are used interchangeably in our work. Mapping mechanisms will decide equivalent and inclusive relations between pairs of ontologies. After that, the mapping results which keep track of semantic equivalence and inclusion concepts are created. Figure 6 is a screenshot of the ontology mapping and a demonstration of the mapping process. The window at the back is an overall prototype when the mapping is selected. The lower left part is a mapping screen where the mapping direction is specified in addition to the two given ontologies. The mapping results in our experiment are shown in the lower right window of Figure 6. The mapping results indicate which concept from one ontology is semantically equivalent to a concept from another ontology after the mapping algorithm has been executed, e.g. $\text{Beer1Agent.beer} = \text{Beer2Agent.suds}$ from the lower right window of Figure 6. Later on when performing ontology integration (Section 6), the integration algorithm knows that Beer1Agent's beer equals to Beer2Agent's suds . Based on these results, further operations such as ontology integration can be achieved.

5.4. Discussion – evaluating ontology mapping

We evaluate the mapping mechanisms in the beer domain with four different beer ontologies (see Section 5.3 above). The evaluation is conducted to verify the quality of the mapping results against the expected results (both equivalence and inclusion defined in Section 3) of the domain. For simplicity, we focus on O'_2

```

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://www.daml.org/2001/03/daml+oil#"
xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
xmlns:gen="http://www.cs.umd.edu/projects/plus/DAML/onts/general1.0.daml#">

<Ontology about="">
<versionInfo>beer-ont, v.1.0</versionInfo>
<comment>An ontology that models brewers and types of beer.</comment>
<imports resource="http://www.cs.umd.edu/projects/plus/DAML/onts/general1.0.daml"/>
</Ontology>

<Class ID="ScotchAle">
<subClassOf resource="#Ale"/>
</Class>
.....

```

(a)

```

beer -- (a general name for alcoholic beverages made by fermenting a cereal (or mixture
of cereals) flavored with hops)
=> draft beer, draught beer -- (beer drawn from a keg)
=> suds -- (a dysphemism for beer (especially for lager that effervesces))
=> lager, lager beer -- (a general term for beer made with bottom fermenting yeast
(usually by decoction mashing); originally it was brewed in March or April and
matured until September)
=> Munich beer, Munchener -- (a dark lager produced in Munich since the 10th
century; has a distinctive taste of malt)
=> bock, bock beer -- (a very strong lager traditionally brewed in the fall and
aged through the winter for consumption in the spring)
=> light beer -- (lager with reduced alcohol content)
.....

```

(b)

```

You'll find more information in the Bierjaarboek 1995-1996.
(Style - Beername, alcohol by volume, Brewery(country)
Aarschots brown ale
Alcohol free beer
Ale
Alt
Amber
Barley wine
Beerette
Bitter
Blending beer
.....

```

(c)

```

All beer can be classified as either a lager or an ale. The differences begin during the brewing
process. Whether the beer is an ale or lager is defined by the type of yeast used in the brew
and the temperature at which fermentation takes place. Ales are brewed with top-fermenting
yeast which allows for rapid fermentation at warmer temperatures. Lagers are brewed with
bottom-fermenting yeast which ferments more slowly and at colder temperatures.
.....
ALE: A top fermented English-style beer. Very few ales are sold commercially in Australia.
The best known are Coopers Sparkling and Pale Ales.

BITTER: In Australia this is likely to be a bottom fermented lager rather than the top-
fermented ale that English would call a bitter. Take XXXX Bitter as an example.
....

```

(d)

Figure 5: Running example ontologies: (a) fragment of beer ontology from DAML ontology library; (b) fragment of beer types from WordNet; (c) fragment of beer types from a Belgian website; (d) fragment of Australian beer types.

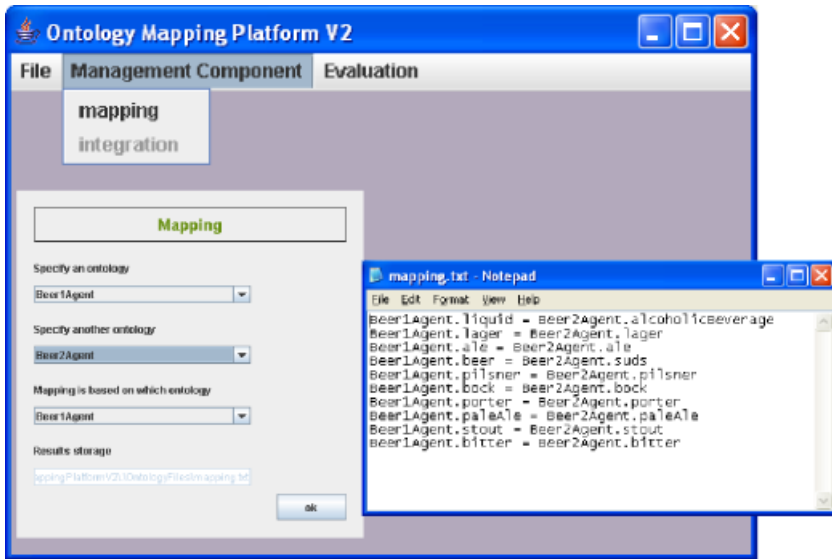


Figure 6: Screen shot of ontology mapping.

(Figure 5(b)), but others can be done in the same way. Notations are defined as follows:

- O_s : source ontology involved in mapping
- O_t : target ontology involved in mapping
- O'_j : ontologies in the running examples of Figure 5 ($j \in [1, \dots, 4]$)
- $M \rightarrow$: two ontologies involved in mapping
- A_Eq**: the number of MAS equivalent relations (by means of the mapping mechanisms proposed in Section 5.2)
- R_Eq**: accurate real number of equivalent relations which should be picked up
- A_In**: the number of MAS inclusive relations (by means of proposed MAS mapping mechanisms)
- R_In**: accurate real number of inclusive relations which should be picked up

Details are displayed in Table 1. As shown, equivalent mapping relations are correctly picked up as seen in columns 4 and 5, but there are differences between columns 6 and 7 in relation to inclusive relations. The difference varies from ontology to ontology. For example, the difference of $O'_4 \rightarrow O'_2$ is 1 compared with 4 for $O'_1 \rightarrow O'_2$.

On one hand, mapping (equivalence) works very well. The reason is that the defined syntac-

Table 1: MAS mapping evaluation

1	2	3	4	5	6	7
O_s	O_t	$M \rightarrow$	A_Eq	R_Eq	A_In	R_In
O'_1	O'_2	$O'_1 \rightarrow O'_2$	10	10	10	6
O'_3	O'_2	$O'_3 \rightarrow O'_2$	5	5	2	0
O'_4	O'_2	$O'_4 \rightarrow O'_2$	7	7	3	2

tic-based similarity is sufficient to deal with the similarity measurement.

On the other hand, mapping (inclusion) works in a slightly different way from mapping (equivalence). This is mainly caused by various conceptualizations of the involved ontologies. Let us take a closer look at the cause.

- $O'_1 \rightarrow O'_2$: in O'_1 , bock, Pilsner, porter and stout are sub-concepts of beer. However, in O'_2 , bock and Pilsner are sub-concepts of lager, while porter and stout are sub-concepts of ale. Also in O'_2 , lager and ale are sub-concepts of suds, which are equivalent to beer according to proposed similarity methods. Consequently, a difference or inconsistency appears. These differences lead to the number of inclusive relations being 10 (Table 1, row 2, column 6) instead of 6 in

reality. More details will be discussed in Section 6.

- $O'_3 \rightarrow O'_2$: for the same reason, porter and bitter with different definitions in O'_3 and O'_2 lead to the number of inclusive relations being 2 (Table 1, row 3, column 6) instead of 0.
- $O'_4 \rightarrow O'_2$: the only difference is bitter. The reason is that in Australia bitter is likely to be a bottom-fermented lager rather than the top-fermented ale that the English would call it. This leads to the number of inclusive relations being 3 (Table 1, row 4, column 6) instead of 2.

In summary, the outcomes of MAS mapping (equivalence) perfectly match the assumed results from human users. However, the outcomes of MAS mapping (inclusion) are not ideal. The differences between pairs of ontologies depend greatly on the actual ontologies, especially their different conceptualizing methods. As we shall see in the next section for ontology integration, consistency checking would exclude the differences shown in columns 6 and 7 of Table 1. In addition, pruning is most likely to cut off inappropriate items in both mappings and integration. The threshold δ was introduced in the mapping mechanism targets to do the pruning. A similar cut-off method will be used in the integration mechanisms as well.

6. Ontology integration

Ontology integration is an important step after the ontology mapping. Ontology interoperability is the outcome when available ontologies comply with ontology mapping and integration. This section addresses the integration process and mechanisms under the proposed framework. This is followed by a prototyping and evaluation discussion.

6.1. Integration process

The vision of MAS would not be realized without agent interactions (Li *et al.*, 2005; Li, 2006). Figure 7 displays the interactions between agents involved in the integration process. In Figure 7, the InA operates directly over the intermediate result, which records all concepts that meet the

requirements, e.g. the items with the number of occurrences greater than a given threshold. A visualization mechanism of the UA can present a graphical view of the result upon request.

In this paper, the integration process starts from the root of a specified ontology and then traverses all sub-concepts. Briefly speaking, the InA counts the appearance of each concept within existing ontologies, and then filters out unexpected concepts with a given threshold. By saying this, we do not mean that we attempt to change the conceptual modelling of the ontology. Rather, ontology integration is based on a specified ontology. The process is described as follows.

- (1) Obtain ontology-related information via OAs (e.g. by accessing the mapping results).
- (2) Keep the numbers of occurrences of each concept in the specified ontology. There are three cases which may take place according to the mapping results.

- *Case 1*: semantic equivalence for the current two concepts (e.g. '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 (e.g. '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 without conflicting with existing sub-concepts of the same ancestor.

- (3) Check the consistency⁵ of the to-be-derived ontology before execution of the insertion operation.
- (4) Filter unexpected concepts by a given threshold.

⁵Consistency checking is aimed at eliminating inconsistency among ontologies, e.g. including the inclusive relations. The details of the checking function are given in the Appendix.

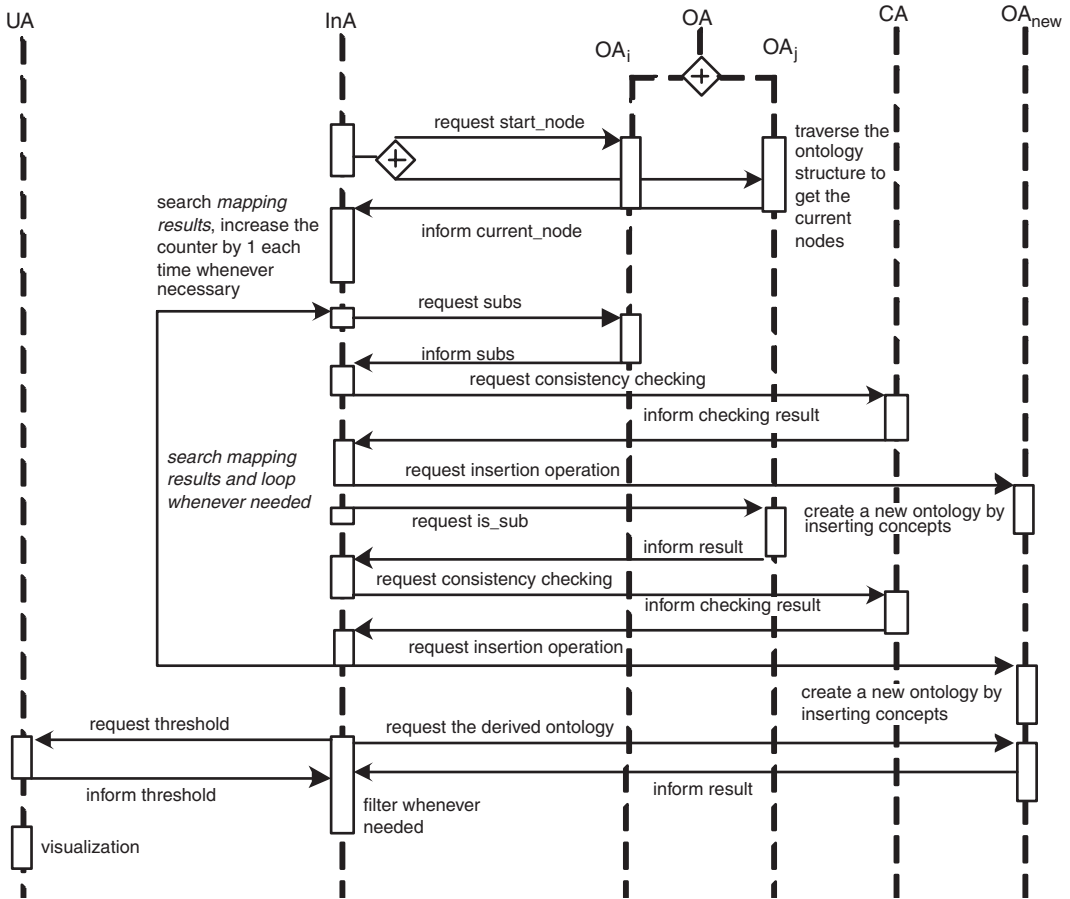


Figure 7: Interactions from InA's view.

The user can request a visualization of the integrated ontology or export this new ontology, e.g. in RDF(s) format.

Following the above approach, a new OA is created instantly (to act on behalf of this ontology) corresponding to the newly derived ontology which is based on the results of the mapping algorithm. The newly created OA enables ontology reuse since OAs in the proposed framework are devised to be responsible for ontology-related tasks. Hence, the integrated ontology can be reused with other existing ontologies in the repository. The presence of the OA makes the integration process self-contained in that it can directly and seamlessly reuse integrated results to continuously improve its effectiveness.

6.2. Integration mechanisms

The integration process operates over the mapping results. The InA is in charge of ontology integration in the environment. The algorithm uses the following functions or data structures to execute relevant operations.

- **initialize:** Initialize the process.
- **next-c:** Request a particular OA for the next concept of a specified ontology. It returns the concept if it exists.
- **search-r:** Search for relations in mapping results. It returns existing relations if they exist or NIL otherwise.
- **copy-c:** Copy a specific concept to the derived ontology properly.

- **insert-super**: Insert a specified concept as a super-node of a given concept in a particular ontology structure.
- **insert-sub**: Insert a specified concept as a sub-node of a given concept in a particular ontology structure.
- **get-threshold**: Contact the UA via the IA to obtain a threshold. It returns the threshold.
- **check-consistency**: Check the consistency of a specific ontology and return a Boolean value to indicate the current status of ontology consistency (details are in the Appendix).
- **filter**: Filter unexpected items from a given ontology, and return the filtered ontology.

The pseudocode of the integration algorithm is shown below. This algorithm may execute

repeatedly until the existing ontologies have been integrated as required.

Pseudocode of integration algorithm

Assume the integration algorithm starts from a given start point.

O_1, O_2 : two different ontologies to be integrated, respectively

O_d : the derived ontology

c_1, c_2 : concepts from O_1 and O_2 , respectively

l_{c_d} : number of occurrences of concepts from O_d

m : a number of available ontologies

relation: relations between given concepts which are from different ontologies

δ : threshold given by the user to filter inappropriate items from the generated ontology

```

1. Function integration {
2. initialise;
3. for ( $i = 1; i < m; i++$ ) {
4.   while ( $(\text{next-c}(O_1) \neq \text{Null}) \ \&\& \ (\text{next-c}(O_2) \neq \text{Null})$ ) {
5.      $c_1 = \text{next-c}(O_1)$ ;
6.      $c_2 = \text{next-c}(O_2)$ ;
7.      $\text{relation} = \text{search-r}(c_1, c_2)$ ;
8.     switch (relation) {
9.       case "=":
10.         $l_{c_d}++$ ;
11.         $\text{copy-c}(c_d)$ ;
12.        break;
13.       case "⊆":
14.        if ( $\text{check-consistency}$ ) {
15.           $\text{insert-super}(c_2, c_1)$ ;
16.           $l_{c_d} = 1$ ;
17.        } //end if;
18.        break;
19.       case "⊇":
20.        if ( $\text{check-consistency}$ ) {
21.           $\text{insert-sub}(c_2, c_1)$ ;
22.           $l_{c_d} = 1$ ;
23.        } //end if;
24.        break;
25.       default:
26.         $l_{c_d} = 1$ 
27.      } //end switch
28.    } //end while
29.  } //end for
30.  $\text{threshold} = \text{get-threshold}$ ;
31.  $\text{filter}(O_d, \delta)$ ;
32. } //end function

```

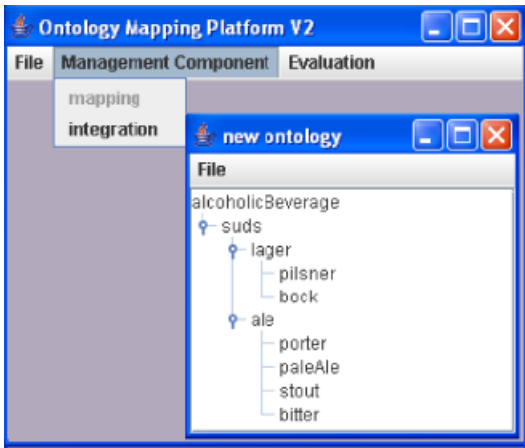


Figure 8: Screen shot of ontology integration.

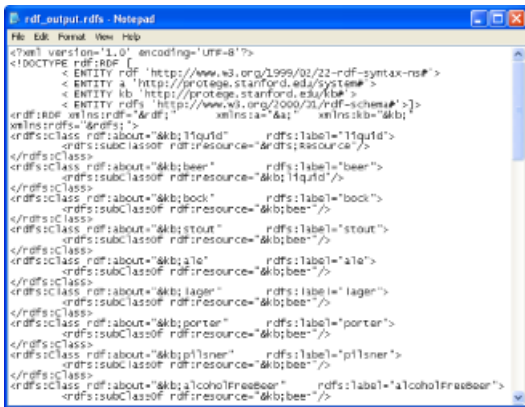


Figure 9: Integrated ontology in RDF(s) format.

6.3. Prototype

Based on the results of ontology mapping, ontology integration aims to formulate a new ontology according to a certain criterion, for instance a threshold δ for the purpose of pruning. Demonstrations have been done on the four running examples (Section 5.3).

Figure 8 is a screen shot of the ontology integration and integration results. The upper left part is the overall prototype when the 'integration' is selected, while the lower right part is the newly integrated ontology in a hierarchical structure. Figure 9 is an RDF(s) format export.

Table 2: MAS integration evaluation

1	2	3	4	5	6	7
Int		A_Eq	A_In	B_ch	A_ch	R_num
O'_1	O'_2	10	10	20	16	16
O'_3	O'_2	5	2	7	5	5
O'_4	O'_2	7	3	10	9	9

6.4. Discussion – evaluating ontology integration

As indicated in Section 5.4, for ontology integration, consistency checking plays an important role in ruling out inconsistency of the newly generated ontology-like representations. Only after existing inconsistencies have been excluded can the integrated ontology then become the one required.

The purpose of consistency checking is to narrow down the number of inclusive relations by using the MAS inclusive method. This will ultimately make it converge on the accurate number of inclusive relations after excluding inconsistencies. For example, the real inclusive relations in $O'_1 \rightarrow O'_2$ are 6 after excluding 4 inconsistencies that were included after ontology mapping (see Table 1). The same is true for other differences presented in the table. Table 2 presents the details of before and after consistency checking against the accurate results. Notations in the table are defined as follows (mapping here includes equivalence and inclusion):

O'_j : ontologies in the running examples of Figure 5 ($j \in [1, \dots, 4]$).

Int: ontologies involved in the integration process

A_Eq: the number of MAS equivalent relations (Section 5.4)

A_In: the number of MAS inclusive relations (Section 5.4)

B_ch: the number of MAS mappings before consistency checking

A_ch: the number of MAS mappings after consistency checking

R_num: real number of mappings

In summary, after mapping and integration, the ontology is accurate for equivalent and inclusive relations.

7. General discussion

With respect to the novelty of our work, we refer back to the problems listed in Section 2 to illustrate its advantages.

7.1. Agent-based framework

Ontologies and ontology-based applications perform in an environment that is dynamic, distributed and heterogeneous. The agent-based framework proposed in this paper is suitable for tasks such as integrating ontologies in a business scenario. By adopting an MAS perspective, interactions among multiple agents are highlighted. This is because they work together to achieve goals beyond their individual capabilities and knowledge. In other words, MASs are able to take a variety of environmental circumstances into consideration rather than treating the environment as being static. Corresponding to Section 2 the evaluation work takes the following characteristics into account (refer to Li (2006) for more details about the design and implementation of the proposed framework).

- *Flexibility*: In the framework, the existing agent communication channel facilitates message delivery. Moreover, the presence of the ontology agents allows for flexible system organization. The system allows ontology agents and functionary agents (including all defined agents for a particular task) to be freely added/deleted to/from the system.
- *Interactivity*: Agents are highly interactive in the framework. Interactions take place not only between ontology agents and functionary agents, but also between functionary agents themselves if a particular task needs to deploy the functionalities of other functionary agents.
- *Interoperability*: The framework enables interoperability between agents of different platforms. In terms of syntactic and semantic heterogeneity of ontologies, a meta-ontology is developed to resolve semantic heterogeneities.

- *Scalability*: In the framework, different classes are developed. They include a *concept class*, an *ontology class* and an *agent class*. Moreover, all ontology-related operations are encapsulated and isolated from the functionary agents' view. By extending corresponding classes, the ontology agent and functionary agent can be created easily.
- *Reusability*: In the framework, whenever a new ontology is generated based on existing ontologies, an ontology agent is created correspondingly. This enables the general view of a particular application domain to be reused in the system.
- *Reliability*: This depends on agents performing rationally in the framework. As every agent in the system is required to register and advertise its capabilities to the interface agent, all other agents are able to reach the available capabilities in the system whenever needed. Moreover, the upper boundary on the number of iterations (the developed agent algorithms) required to reach a fixed point is the number of concepts in an ontology. It is known that the number of concepts in an ontology is finite.

7.2. Prior knowledge

As soon as the problems behind ontology management were recognized, the limitations of having predefined prior knowledge to support it began to be manifest. Despite the fact that prior knowledge in a system cannot be totally ruled out, little prior knowledge is becoming possible as agents within a system are able to obtain necessary knowledge through interaction. As a result, deploying agent technologies in ontology management as described in this paper is most probably able to accommodate the changes of a hosting environment.

7.3. Ontologies

In the beginning, dynamic ontology management for the purpose of providing interoperabil-

ity among ontology-based applications was not a focus of research. Gradually, interest in tackling ontology interoperability issues, especially in the real world, has started to bloom. It seems that a flexible and effective way to tackle ontology research is necessary since more and more ontologies from different organizations come into applications. Our concentration is significantly different from others and we have emphasized the dynamic change of ontologies and made a good effort to cope with this.

We have evaluated the mapping and integration methods in the beer domain, where four different ontologies are used for illustration. Our methods (mapping and integration) seem to be generic which implies that they are applicable across different domains.

8. Related work

Limited work has been done regarding ontology integration, so existing work is mainly concentrated on ontology mapping with little concern about integration. Current work in this area falls roughly into three groups. The first is about using specific techniques or knowledge to facilitate ontology mapping, especially in similarity measurement definitions. Examples are GLUE (machine learning techniques) (Doan *et al.*, 2003), RDFDiff (change-classification rules) (jointly developed by VU Amsterdam and Ontotext Lab), FCA-Merge (formal concept analysis) (Stumme & Maedche, 2001), ONION (articulation rules) (Mitra *et al.*, 2000; Mitra & Wiederhold, 2001), MAFRA (semantic bridge) (Maedche *et al.*, 2002; Silva & Rocha, 2003), and an integrated approach (Ehrig & Sure, 2004). The second attempts to provide tools for designers/users to facilitate ontology mappings, e.g. the PROMPT (to suggest to designers which concepts may be related) (Noy & Musen, 2003) and OntoMap (to compare ontologies) (Kirya-kov *et al.*, 2001). The third introduces architectures as well as matching approaches, e.g. OBSERVER (Mena *et al.*, 2000), MOMIS (Bergamaschi *et al.*, 1999, 2001) and InfoSleuth (Fowler *et al.*, 1999; Nodine *et al.*, 2000). Of all

this related work, only InfoSleuth is an agent-based system developed to synthesize new technologies into a unified system that retrieves information with changes of information sources. However, it significantly lacks capabilities such as brokering and monitoring capabilities which are strongly supported in the latest developed agent technical platform, e.g. Jade. Moreover, it does not comply to FIPA specifications. On the other hand, there is no automation support in creating mappings between ontologies and data schemas in InfoSleuth. In addition to InfoSleuth, the earlier OBSERVER project attempted to aid users to observe a semantic conceptual view of a global information system by giving the user the ability to browse multiple domain-specific ontologies. In contrast to automatically creating the mappings between ontologies, OBSERVER used a predefined Inter-ontology Relationship Manager to provide translations between the terms of different ontologies.

In short, although the architecture is key for ontology management in a dynamic environment, it has not yet been thoroughly investigated. Rather than developing the system from scratch, by taking advantage of both the *agent management system* (AMS) and the *directory facilitator* (DF), together with other tools (e.g. *Remote Management Agent*) bundled with Jade to simplify administration and application development, we concentrate on the overall framework and functionalities of every kind of agent (including mapping and integration mechanisms). Our proposed framework aims to provide flexibility and extensibility in dealing with ontology mapping and integration in a dynamic and heterogeneous environment.

9. Conclusion and future work

In this paper, we have discussed a new agent-based framework for ontology-based applications among distributed and heterogeneous ontologies. The novelty of our work is not solely in the use of agent technology but rather in the way it facilitates the synergy of MASs for dynamic ontology management. We argue that

the agent-based approach enhances ontology interoperability in which it overcomes some limitations of existing systems and tools.

In this paper, we have also discussed the corresponding mechanisms for ontology mapping and integration in MASs. This is realized by using the Jade agent platform. Each kind of agent has been detailed and implemented within the proposed framework. We have illustrated how ontology mapping and integration function in the prototype. More importantly, we have evaluated the results which show the effectiveness of the work. The evaluation work derives valuable conclusions on which further work can be based.

We are extending the capabilities of functionary agents by creating more agents to play roles such as planning. There are some other directions for future work. Among them, the semantics match by considering axioms of an ontology is one of the toughest questions. We intend to investigate ontology mapping from an algebraic perspective by treating an ontology as an abstraction in the sense that it can represent any relations with complex structures (axioms). We hope that an algebraic abstraction will allow us to deal with ontology mapping in the same way as well-known computational metaphors. Furthermore, we are considering other possible relationships between two concepts in addition to taxonomic relations. We also plan to use a process algebra to support agent interactions at a high level of abstraction. We hope that this perspective will allow the proposed framework to deal with tasks among agents in abstract yet more flexible ways.

Acknowledgement

Work reported here is partly supported by Swinburne Vice Chancellor's Strategic Research Initiative Grant. The authors are grateful for the constructive comments of the anonymous reviewers and the prototyping work of Shane Grund.

Appendix: Consistency checking function

The consistency checking function operates over a single ontology to check whether the ontology is consistent or not. The CA is in charge of ontology consistency checking in the environment. The algorithm uses the following functions or data structures to execute relevant operations.

- **initialize**: Initialize the process.
- **get-super**: Obtain the super-node of the current concept.
- **get-in**: Obtain an inclusive relation from the mapping results (discussed in Section 5.2).
- **get-eq**: Obtain an equivalent relation from the mapping results.
- **search-c**: Return 'true' if the concept is in a data repository, otherwise 'false'.
- **search-r**: Search for relations in mapping results. It returns existing relations if they exist or NIL otherwise.
- **attach-sub**: Attach a concept as a sub-node to a given concept.

The pseudocode of the consistency checking algorithm is shown below. This algorithm executes repeatedly until all super-nodes have been checked.

Pseudocode of consistency checking

Assume the original ontology is consistent. The checking algorithm starts when two ontologies are to be integrated, especially in the presence of inclusive relations.

c_1, c_2, c'_2 : concepts – it is worth noting that these concepts may come from different ontologies.

list: a data repository to record all concepts included in the current ontology.

relation: relations between given concepts which are from different ontologies.

ck: a flag with a Boolean value.

comp-attach: a sub-function with details described below.

```

1. Function comp-attach ( $c_1, c_2, c'_2$ ) {
2.   while (get-super( $c_1$ )! = Null) && (get-super( $c'_2$ )! = Null) {
3.     if !(get-super( $c_1$ ))  $\cap$  get - super ( $c'_2$ ) {
4.        $ck = false$ ;
5.       return;
6.     } //end if;
7.      $c_1 = \text{get-super}(c_1)$ ;
8.      $c'_2 = \text{get-super}(c'_2)$ ;
9.   } end while
10.    $ck = true$ ;
11.    $relation = \text{search-r}(c_1, c_2)$ ;
12.   switch( $relation$ ) {
13.     case " $\supseteq$ ":
14.       attach-sub( $c_1, c'_2$ );
15.       break;
16.     case " $\sqsubseteq$ ": {
17.       attach-sub( $c'_2, \text{get-super}(c_1)$ );
18.       attach-sub( $c_1, c'_2$ );
19.     } break;
20.   } end switch
21. } //end function

1. Function check-consistency {
2.   initialise;
3.   get-in( $c_1, c_2$ );
4.   if !(search-c( $list, c_2$ )) {
5.     get-eq( $c_2, c'_2$ );
6.     if (search-c( $list, c'_2$ )) {
7.       comp-attach( $c_1, c_2, c'_2$ );
8.     } // end if
9.   } else {
10.    comp-attach( $c_1, c_2, c_2$ );
11.  } //end if
12. } //end function

```

References

- BAUER, B., J.P. MÜLLER and J. ODELL (2001) Agent UML: a formalism for specifying multiagent interaction, *International Journal of Software Engineering and Knowledge Engineering*, **11** (3), 207–230.
- BERGAMASCHI, S., S. CASTANO and M. VINCINI (1999) Semantic integration of semistructured and structured data sources, Special Issue on Semantic Interoperability in Global Information, *SIGMOD Record*, **28** (1), 54–59.
- BERGAMASCHI, S., S. CASTANO, D. BENEVENTANO and M. VINCINI (2001) Semantic integration of heterogeneous information sources, Special Issue on Intelligent Information Integration, *Data and Knowledge Engineering*, **36** (3), 215–249.
- BERNERS-LEE, T., J. HENDLER and O. LASSILA (2001) The semantic web, *Scientific American*, **284** (5), 34–43.
- BYRNE, J.A., R. BRANDT and O. BORT (1993) The virtual corporation, *Business Week*, **8** (February), 36–40.
- DOAN, A., J. MADHAVAN, R. DHAMANKAR, P. DOMINGOS and A. HALEVY (2003) Learning to match ontologies on the semantic web, Special Issue on the Semantic Web, *VLDB Journal*, **12** (4), 303–319.
- EHRIG, M. and Y. SURE (2004) Ontology mapping – an integrated approach, in *Proceedings of the 1st Eur-*

- opean Semantic Web Symposium, C. Bussler, J. Davis, D. Fensel and R. Studer (eds), LNCS 3053, Berlin: Springer, 76–91.
- FOWLER, J., M.B. NODINE, B. PERRY and B. BARGMEYER (1999) Agent based semantic interoperability in InfoSleuth, *SIGMOD Record*, **28** (1), 60–67.
- GRUBER, T.R. (1993a) A translation approach to portable ontology specifications, *Knowledge Acquisition*, **5** (2), 199–220.
- GRUBER, T.R. (1993b) Toward principles for the design of ontologies used for knowledge sharing, *KSL-93-04*, Knowledge Systems Laboratory, Stanford University; <http://ksl-web.stanford.edu/>.
- JENNINGS, N. (2000) On agent-based software engineering, *Artificial Intelligence*, **117**, 277–296.
- JENNINGS, N. and M. WOOLDRIDGE (2001) Agent-oriented software engineering, in *Handbook of Agent Technology*, J. Bradshaw (ed.), Boston, MA: AAAI/MIT Press.
- KIRYAKOV, A., K. SIMOV and M. DIMITROV (2001) Ontomap: the portal for upper level ontologies, in *Proceedings of Formal Ontology in Information Systems*; available from <http://www.ontotext.com/publications/index.html>.
- LENAT, B.D. and V.R. GUHA (1990) *Building Large Knowledge-Based Systems, Representation and Inference in the CYC Project*, Reading, MA: Addison-Wesley.
- LI, L. (2006) Agent-based ontology management towards interoperability, PhD Thesis, Swinburne University of Technology, Australia; <http://www.it.swin.edu.au/personal/lli/thesis.pdf>.
- LI, L., Y. YANG and B. WU (2005) Agent-based ontology integration for ontology-based application, in *Proceedings of the Australasian Ontology Workshop*, T. Meyer and M. Orgun (eds), Conference in Research and Practice in Information Technology (CRPIT) Series 58, Sydney: Australian Computer Society, 53–59.
- LUCK, M., P. MCBURNEY and C. PREIST (2004) A manifesto for agent technology: towards next generation computing, *Autonomous Agents and Multi-Agent Systems*, **9** (3), 203–252.
- MAEDCHE, A., B. MOTIK, N. SILVA and R. VOLZ (2002) MAFRA – Mapping FRAMework for distributed ontologies, in *Proceedings of the 13th International Conference of Knowledge Engineering and Knowledge Management*, A. Gómez-Pérez and V.R. Benjamins (eds), LNCS 2473, Berlin: Springer, 235–250.
- MENA, E., A. ILLARRAMENDI, V. KASHYAP and A. SHETH (2000) OBSERVER: an approach for query processing in global information systems based on interoperation across pre-existing ontologies, *Distributed and Parallel Databases*, **8** (2), 223–271.
- MITRA, P. and G. WIEDERHOLD (2001) An algebra for semantic interoperability of information sources, *IEEE International Conference on Bioinformatics and Biomedical Engineering*, New York: IEEE Press, 174–182.
- MITRA, P., G. WIEDERHOLD and M. KERSTEN (2000) A graph-oriented model for articulation of ontology interdependencies, *Proceedings of Conference on Extending Database Technology*, pp. 59–69.
- NODINE, M., J. FOWLER, T. KSIEZYK, B. PERRY, M. TAYLOR and A. UNRUH (2000) Active information gathering in InfoSleuth, *International Journal of Cooperative Information Systems*, **9** (1–2), 3–28.
- NOY, N.F. and M.A. MUSEN (2003) The PROMPT suite: interactive tools for ontology merging and mapping, *International Journal of Human–Computer Studies*, **59** (6), 983–1024.
- SILVA, N. and J. ROCHA (2003) Ontology mapping for interoperability in semantic web, in *Proceedings of the IADIS International Conference WWW/Internet 2003*, pp. 603–610.
- STUMME, G. and A. MAEDCHE (2001) Ontology merging for federated ontologies on the semantic web, in *Proceedings of the International Workshop for Foundations of Models for Information Integration*, Berlin: Springer, 16–18.
- WOOLDRIDGE, M. (1997) Agent-based software engineering, *IEE Proceedings — Software*, **144** (1), 26–37.
- WOOLDRIDGE, M. (2002) *An Introduction to Multi-Agent Systems*, New York: Wiley.

The authors

Li Li

Li Li received her PhD degree with major in computing from Swinburne University of Technology, Melbourne, in 2006. She has published more than 30 papers in journals and refereed conferences. Her research interests include service-oriented computing, semantic web, ontology engineering and multi-agent systems.

Yun Yang

Yun Yang is currently a full Professor in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne. Prior to joining Swinburne as an Associate Professor, he was a Lecturer and Senior Lecturer at Deakin University during

1996–1999. Before that, he was a (Senior) Research Scientist at DSTC Cooperative Research Centre for Distributed Systems Technology during 1993–1996. He also worked at the Beijing University of Aeronautics and Astronautics during 1987–1988. He has edited several books and

published more than 160 papers in journals and refereed conferences. His research interests include software technologies, p2p and grid workflow systems, service-oriented computing, Internet computing applications, computer-supported collaborative work, and e-business processes.