

# An Infrastructure for Agent Collaboration in Open Environments

Kenichi Yoshimura<sup>1</sup>, Lin Padgham<sup>1</sup>, and Wei Liu<sup>2</sup>

<sup>1</sup> School of Computer Science and Information Technology,  
RMIT University, Australia  
{kenichi,linpa}@cs.rmit.edu.au

<sup>2</sup> School of Computer Science and Software Engineering,  
The University of Western Australia, Australia  
wei@csse.uwa.edu.au

**Abstract.** In this paper, we present a service composition tool which enables specification of composite services for users with limited programming skills. We view composite services as a loose form of *teamwork* of service provider agents coordinated by a personal user agent. Such view enables us to use an existing hierarchical model of teamwork as the basis for service composition. An extension to the current model of teamwork is also proposed to achieve *flexible* and *robust* execution of composite services in an open environment.

**Keywords:** Intelligent Agents

## 1 Introduction

The World Wide Web is undergoing a significant evolution since Berners-Lee et al. unleashed the semantic web possibilities in 2001 [2]. The ultimate aim of the semantic web is to revolutionise today's web (a human friendly and dependent information repository) into a machine (agent) friendly backbone connecting automated service providers across the Internet.

The existence of content markup and service description languages (e.g. DAML-S and WSDL [12]) alone is not enough to realize the semantic web vision. Many have identified the importance of *service compositions* [5, 13] for developing more complex services using existing services available on the network.

Agents and agent technologies are well recognized as the key players and the active components in the semantic web world [2]. They are to interpret, manipulate and execute web services in an open environment.

Viewing web services as agent services, as far as we are concerned, a service composition problem naturally turns into an agent coordination and collaboration issue. In this paper, we present a novel infrastructure that uses teamwork framework to realize *flexible* and *robust* service composition.

It is well known that the success of the current Web attributes to its ease of use and the fault tolerance nature of web browsers. If the Semantic Web is to enjoy a similar level of acceptance by the large majority of normal end users,

intuitive service composition tools should be developed to enable users browsing and combining services in a familiar way as browsing and using the Web content today. We have implemented a prototype system in order to explore such vision.

The Web is an *open environment* where services are deployed by different organizations using various technologies. One of the key characteristics of an open environment is that services are not available reliably all the time. The system evolves over time and new services may be added at an ongoing basis. A successful system operating in such environment must be both *flexible* and *robust* to cope with the uncertain nature of the environment.

Previous study of Beliefs, Desires, and Intentions (BDI) architecture has shown it is one of most successfully implemented platforms (e.g. JACK [4] and JAM [9]) for developing systems in dynamic environment [7, 14]. In addition, work on *teamwork* addresses issues of how to improve robustness of systems consist of numerous agents (discussed further in Section 4). For this reason, we use a BDI architecture, JACK agent development framework, which also provides an extension for teamwork for the deployed prototype system. We present a generalizable extension of such a teamwork model that is particularly suitable for representing composite services.

Our prototype system is built for, and deployed in, the Agentcities<sup>3</sup> network. Agentcities is an international initiative, which aims at facilitating the exploration of a worldwide open network of platforms hosting a variety of interoperable agent services. Agentcities grew out of the work of the Foundation for Intelligent Physical Agents (FIPA)<sup>4</sup>, which is a standards body focussing on standards and infrastructure for rational interaction and cooperation among heterogenous agents.

In this paper, we present:

- A novel view of service composition as a loose form of teamwork, and an exploration of a team programming environment as a base for service composition;
- The use of a goal directed BDI framework to assist in achieving robustness;
- Development of mechanisms for repairing a plan in the event of a team member (i.e. a service provider agent) becoming incapacitated or failing to provide the needed service.

In the following sections we present a general approach for realizing our vision of a service composition tool and explore key characteristics of the design of our prototype system. It is known as **ASKIT**, **A**gent **S**ervice **K**omposition **I**nterface **T**ool<sup>5</sup>. In section 2, an overall architecture is described. Section 3 introduces the key concepts of a BDI team oriented programming environment, JACK Teams. Section 4 explores the team view of service composition, followed by Section 5 describing our extension to the current team plan model. Section 6 compares our work with others and the paper concludes in section 7.

---

<sup>3</sup> <http://www.agentcities.org>

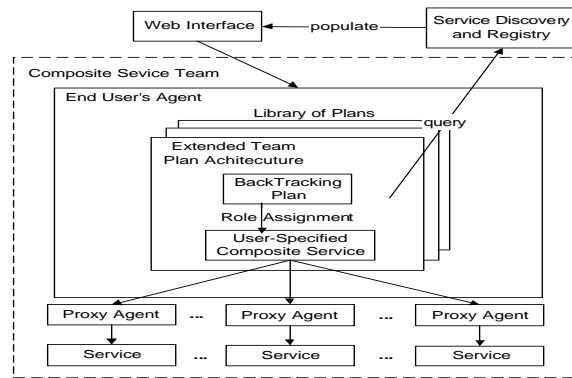
<sup>4</sup> <http://www.fipa.org>

<sup>5</sup> The prototype of ASKIT for demonstration purposes is available at <http://agentcities.cs.rmit.edu.au:7198/agentcities/login.jsp>.

## 2 An Overall Architecture

Figure 1 depicts the key components and data flow of an infrastructure that supports automatic FIPA compliant agent service composition. It consists of three core components:

- *Web interface* is responsible for collecting data from end users.
- *Service discovery and registry module* keeps service provider details, service types and service APIs. It is also responsible for populating the Web interface and display available service APIs in an intuitive manner to the end user.
- *Team plan generation module* is responsible for generating abstract composite service specification from the Web interface data, generate abstract team plans, backtracking upon plan failures, and generating *proxy agents* for interacting with service providers.



**Fig. 1.** A service composition Infrastructure

In the context of research presented in this paper, a service is modelled as a service type and a list of interfaces associated with the service. Each interface has a list of expected input parameters and a list of expected output values. For example, a bank service type has interfaces to open an account, it takes your name, address and passport number as input values, and it returns your account number as an output value of the interface<sup>6</sup>. Multiple instances of the same type of service can be available on the network simultaneously. For example, it is possible that several CD seller services are available on the network, offering competitive prices. This is one of the key characteristics of the open environments where services may appear or disappear by the interest of service providers, and services are not reliably available all the time.

<sup>6</sup> Due to the limit of pages, interested user are referred to the ASKIT Web site for further details.

The interface also allows a user to specify the control sequence of a composite service by combining the existing services sequentially, or in parallel, or combination of both.

One of the key characteristics of our architecture is composite services is specified using *service types* rather than *service providers*. When composite services are executed, the infrastructure identifies appropriate service providers depending on the configuration of the environment. This enables us to implement a flexible and adaptive service composition tool which in particular suits for the dynamic and evolving Web environment.

The current model of services makes simplifying assumptions about service description and Ontology issues. Currently, the system only interacts with a known set of services using directory facilitators for *agent service discovery*. However, this should not influence the demonstration of our service composition ideas. We expect to extend the Service Discovery and Registry module to accommodate the current industry standard such as UDDI<sup>7</sup>.

The infrastructure automatically generates the source code for a team *proxy agents* that are responsible for interacting with each service provider agent so as to fulfill the specified composite service. This is based on our view of composite services as *teamwork* among service provider agents. Section 4 has further discussion on this, and Section 5 describes how the current model of teamwork can be extended to improve the overall flexibility and robustness of the system.

### 3 JACK Teams

JACK Intelligent Agents<sup>TM</sup> is an extension of the Java programming language designed to provide additional programming constructs for developing applications<sup>8</sup>. It is based on the Beliefs, Desires, and Intentions model [3] and previous practical implementations of such systems, for example, JAM [9].

The BDI agent model is an event-driven execution model providing both reactive and proactive behaviour. In this model, an agent has certain beliefs about the environment, has goals (desires) to achieve, and has plans (intentions) describing how to achieve goals. One of the key strengths of the BDI architectures is their ability to recover from failures by selecting alternative plans to achieve the same goal.

JACK Teams is an extension of JACK Intelligent Agent<sup>TM</sup> for *Team Oriented Programming*. It is a nuance of *Agent Oriented Programming* aimed at developing teams of agents. Programming of teamwork amongst agents is an area that has been recognised as causing a range of challenges and difficulties.

JACK Teams does not operate with the traditional constructs of mutual belief, joint goals, and joint intentions [6]. Rather it has a hierarchical team model where the team entity at each level is reified and contains its own beliefs, goals and intentions, giving a model of team. JACK Teams takes the approach

---

<sup>7</sup> Universal Description, Discovery and Integration of Web Services at <http://www.uddi.org>

<sup>8</sup> An evaluation package is available for download from [www.agent-software.com.au](http://www.agent-software.com.au)

of modelling the team as a specific entity, separate from its members. Having a team entity explicitly modelled allows for reasoning to be done at the level of the team, rather than at the level of team members. Thus reasoning about which team plan to use, based on the situation, is encapsulated within the team entity.

Roles are conceptual entities which are defined by the events that are required to be handled by that role, events that can be generated by that role - with the expectation that the team entity will handle these and beliefs which are propagated to and from the role.

A team plan is used to describe how to go about achieving a particular goal of the team executing the plan. A team plan declaration includes a set of team members required to achieve its goal (called *role positions*). The strategy within a team plan typically involves co-ordination of its team members. It is described in terms of *role positions* associated with the team plan using programming constructs such as *@team\_achieve(...)* (sets out a goal for a team member) and *@parallel(...)* (executes multiple tasks concurrently) statements.

A team entity declaration includes what roles the team uses. Once a team entity is instantiated, it is possible to assign members to *role containers* which can be thought of as groupings of team members, organised according to the kind of function, i.e. role, they can have within the particular team. At this stage there is no commitment for any particular team member to play a part within a given team plan. Once the team entity chooses a plan to execute in the service of one of its goals, the role-positions required by the plan must be filled by particular agents from the role containers. It is only when a member accepts a role position in a particular plan that it commits to acting on behalf of the team.

JACK Teams is especially well suited to modelling larger organisations where there is a hierarchical structure which can readily be modelled as teams within teams. In these contexts the concept of the team as an entity which can itself be a team member is very powerful.

## 4 Composite Services and Teamwork

In this section, we present our approach of viewing composite services as a form of *teamwork*. A comparison study is also presented to differentiate our approach from the existing teamwork literature.

### 4.1 A Team Plan View of Service Composition

In the work presented in this paper, we view a composite service as a particular form of hierarchical *teamwork* involving an agent pursuing goals on behalf of its user and service providers. In this model, only the user agent views its service providers as its team members, and the service providers do not have any awareness of their roles played within the team.

The standard view of agent interactions in a system such as Agentcities is that interaction between agents happens according to protocols which are agreed in advance and which form the basis of interaction patterns between participating agents. One of the difficulties of the protocol based view of agent interactions is that the protocols must be decided in advance, prior to the development of all agents who will use these protocols. In this context, we refer to an interaction protocol as a description of exact sequences of permitted messages between the interacting agents (service providers). It is a useful implementation level description of interactions, and developers of the system should follow the specification. It is important to note that such a specification includes all possible scenarios of the conversation including successful and failure cases of the interaction.

Our view is that while some basic protocols are necessary at the implementation level, it is useful to be able to specify and program customized and extended conversation using the existing services without detailed specification of individual conversation. That is, such a specification of composite services should only capture the new scenarios of composite services without showing every possible conversation with every individual services. In our approach, the translation process of such a specification of composite services deals with the detailed specification of individual conversations.

This is one of key characteristics of our system which enables the end users with limited programming experience to compose new services using the services available on the network. From the end users' perspective of our system, the specification of a composite service is only a matter of specifying synchronization and co-ordination of existing services and input values for each service.

Such an abstract representation of composite services captures the aspects of a composite service that are concerned with the co-ordination and synchronization of messages between the service providers and the initiating agent. Team plans as described in the previous section are concerned with synchronization and co-ordination of activities done by different agents. Consequently we see team plans as an ideal tool to use for both coordinating the actions and interactions of agents and for composing the services offered by other agents.

In JACK Teams, the separation of *teams* from individual agents enables an abstract specification of coordination strategies as team plans. An individual agent then figures out how to achieve its assigned task from the team and reports the outcome of the assigned task once it finishes the execution. In fact, such an approach of teamwork model enables an implementation of service composition model which we proposed above.

In this model, team plans are used to capture the order of execution of composite services. It only captures the sequence of interaction which the end user of our system specified. An individual interaction with a particular service provider is represented as a task for a team member to achieve, and it is the team member's responsibility to ensure coherent interaction with the service provider.

## 4.2 Roles of Proxy Agents

A service composition involves more than a simple exchange of messages between service providers and a user agent. For example, an interaction involves discovery of service providers, management of different interaction protocols, and message delivery over the network just to name a few issues involved. Furthermore, service providers are not aware of their roles within a team. For example, a service provider could disappear any time during the interaction, and it does not agree to meet any responsibilities as a member of a team.

*Proxy agents* are responsible for the management of individual interaction with service providers. Proxy agents hide most of the environment specific requirements (FIPA standards in this case) and ensure reliable interaction with unreliable services. Key roles of a proxy agent are summarized below.

- Failure Management: *Proxy agents* deal with failure of various kinds, eg. a restaurant booking service is unable to make a booking due to its restaurant is fully booked.
- Time out management: An *proxy agent* monitors the progress of an interaction. It declares failure if the service provider associated with the proxy agent does not reply within a reasonable time.
- Interaction protocol management: Currently each proxy agent knows *FIPA-QUERY* and *FIPA-REQUEST* interaction protocols. Depending on the interaction protocol used by a service provider, the proxy agent prepares appropriate messages.
- Content language management: Conversion of content language to/from an end user friendly format.
- Message delivery management: Delivers messages over the network and reports any failures.

The proxy agents implement the basic requirements to play a part in a team, and it enables our view of composite service as a form of teamwork transparently to service providers.

## 4.3 A Comparison with the Teamwork Literature

Over the past decade, theoretical analyses of teamwork [6, 8, 11] have lead to a variety of practical approaches to facilitating the implementation of coordinated agent activity, e.g. [1, 16]. It is aimed at decreasing the complexity of developing a team of collaborative agents and improve the overall robustness of the system. Aspects of this have been well illustrated by empirical work, e.g. [16].

In the teamwork literature, collaborative activities are typically achieved by team members having certain beliefs about the team while executing team plans. For example, [6] argue that teamwork involves more than team members obeying certain rules, but also team members having certain commitments as a member of the team such as being responsible to make its knowledge about the progress of the current goal mutually known when it knows something different from what was known previously.

In comparison with the previous work on teamwork, our view of composite services as a form of teamwork has a much looser notion of *teams*. Our approach realizes that a service composition is analogous to a specification of team plan from the perspective of a user who is composing the services. Furthermore, the implementation of the service composition infrastructure based on the teamwork framework would complement each other since both domains inherently share the similarities.

Despite the fact that our approach does not comply with the previous work on teamwork in several aspects, we can assume the outcomes of co-operative tasks, i.e. composite services, will not suffer from constant failures due to lack of understanding as a team amongst team members. It is based on the assumption that service providers are typically co-operative once they agree to provide services although the services can be expensive <sup>9</sup>.

In the service composition domain, we believe a reliable discovery of services and a smart recovery strategy from failed interaction are important to achieve the robust execution of composite services. Section 5 describes our approach to improve the current model of teamwork to address these issues.

## 5 Extending Team Plans for Robust Execution

In any form of teamwork, an outcome of joint effort largely depends on the participating team members' performance as well as the environmental influences. For example, one of the major motivations of Tambe's work [16] on his general model of teamwork is to improve the overall robustness of executing joint plans by having generalised model of teamwork as a part of the underlying infrastructure. A large portion of the underlying teamwork model is concerned with the behaviours of failed team members. Furthermore, additional work has been done to improve the robustness by exploring the possibilities of monitoring the progress of team members [10].

The uncertain nature of the open environments introduce a new problem to the current teamwork models which has not been addressed in the current practical work on BDI platforms and teamwork models. Typically, the failure recovery approach available in such systems is the infrastructure support to select an alternative plan to achieve the same goal. This process continues until either the goal is achieved or all applicable plans failed to achieve the goal. Such an approach to failure recovery potentially imposes computational redundancy especially in the service composition type of teamwork. For example, unless a smart team member selection algorithm is implemented in the system, an alternatively selected plan could select a completely different set of team members and the effort of previously selected team members is completely wasted. In other words, in face of a certain sub-team failure, we do not want to reconstruct the whole team again, we want to be able to replace only the failed sub-team or team member.

---

<sup>9</sup> We are currently not concerned with the security issues of open environments.

We propose a new approach of failure recovery which extends the current model of teamwork as an additional means to achieve the robust execution of team plans. In this model, a failed plan is analyzed to identify the cause of failure, finds a replacing team member for the failed step, performs *backtracking* and the failed plan resumes with a new set of team members.

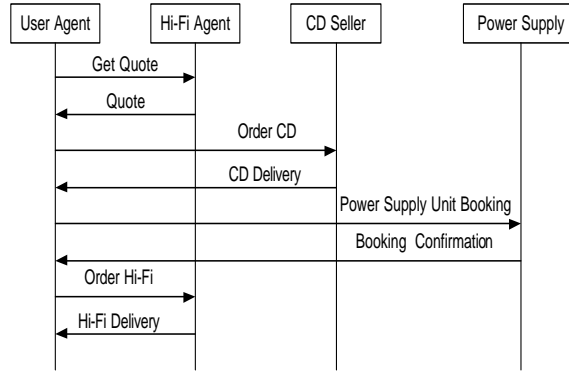
This approach is based on our observation of composite services in open environments where the failure of a particular service does not often indicate the failure of the entire composite service. This approach is suitable given the fact that there are multiple instances of the same type of service available, and services are typically independent of each other since they are deployed by different organizations. Also bear in mind, we are trying to develop an easy to use service composition tool for end users who have limited programming experiences. Therefore, it is more desirable to ensure the robust execution of specified composite services, rather than relying upon the users to specify alternative plans to achieve the same goal.

In our approach, a generated team plan is executed until either the plan succeeds, or all combination of team members (service providers) are attempted and have failed to fulfil the requirements. This is an additional level of *backtracking* we built on the existing BDI model of failure recovery. The backtracking process involves five steps:

1. Find a failed step in the team plan
2. Find an alternative service provider of the same type and replace the failed service provider
3. Mark all the interfaces associated with the failed service provider as *not executed*
4. Identify any other steps which depend on the output of failed steps and mark as *not executed* (perform this recursively)
5. Re-execute the plan

To implement the *backtracking* strategy, we implemented an additional data structure which keeps track of the current execution state of a team plan. The data structure holds information required to perform the *backtracking*, which includes information such as an *execution-state flag* (true if it was successfully executed), *dependency with other steps* in the plan, and *assignment of a team member to a role*. Dependencies amongst the steps of a team plan are analyzed at the compilation time of the user specified composite services.

A hypothetical example illustrating the need for smart backtracking is a situation where a user wishes to purchase a hi-fi unit for his office, and a CD, but also wants to check that the hi-fi unit chosen is one for which there is a suitable power supply amongst those available at the workplace. Assume there are agents selling hi-fi units, an information agent providing information about available power supply units and agents that sell CDs. It is necessary to select a hi-fi unit in order to check availability of the power supply, as this is different for different units. However the user does not want to commit to purchase until it is confirmed that an appropriate power supply is available. Figure 2 illustrates the interaction.



**Fig. 2.** Hi-Fi Ordering Example

Assume that the first three steps execute successfully, but the fourth step fails as the agent has now sold the hi-fi that was available. This necessitates going back to redo the first step. The third step will also need to be re-done, as it is dependent on the output of the first step. The second step on the other hand is independent and does not need to be repeated. Our system takes care of these dependencies, backtracking to redo only necessary steps.

The additional level of *backtracking* we developed could easily be extended to implement a generalized model for other domains. Our approach potentially provide a useful basis for development of multi-agent systems, in particular where a selection of team members is important.

## 6 Related Work

In this section, we discuss three pieces of related work that are strongly relevant to our work presented in this paper.

*eFlow* [5] is a service composition platform designed for specification, management, and execution of composite services using the existing Web-based services, primarily designed for the future *e-business* applications. The project aims at developing an enabling technology for more customisable and dynamic composite services. A composite service is modeled as a graph which defines the order of execution. The nodes in a graph includes service selection rules, which enables a dynamic selection of service providers. The system is flexible to incorporate new services and *broker agents* as the configuration of environment changes over the time.

*eFlow* and the ASKIT share great deal of similarities. In particular, both approaches enable specification of sequential and parallel execution, dynamic allocation of service providers, and provides support to access every instances of the same type of service just to name a few common characteristics. *eFlow*

focuses on the development of infrastructure support for organization deploying services and manage their services easily. In comparison, our project aims at developing a service composition tool that enables end users with limited programming experiences compose *flexible* and *robust* services using services available on the network.

*SWORD* [13] is another implemented service composition tool. In *SWORD*, a service is represented as a simple rule using its expected input values and output values. Those rules of existing services are stored in a rule-based expert system and they are used to determine if a new composite service can be realized using the existing services given the expected input and output values of the new service. The primary contribution of their work is their approach of automatically generating new composite services. Our approach differs significantly in that our interface enables a user compose customized services by directly manipulating a set of service types currently available on the network. Ultimately, these user specified composite service can be kept in plan libraries for re-use.

Sycara et. al. [15] present an agent capability description language called *LARKS* that is used for the *matchmaking* process of services available on the Internet. Their work is primarily concerned with the development of a service description language and the implementation of the efficient and adaptive match-making process. Work on service description languages such as *LARKS* and *WSDL* is important for the service registration and discovery aspect of our system.

## 7 Conclusions

In this paper, we present a service composition tool-*ASKIT*, which enables users with limited programming experiences to compose customized services using the existing services available on the Agentcities network. We extended the current teamwork model in that a failed team plan is repaired by only replacing the failed team member, i.e. service provider, but not discard the whole team. *Backtracking* is performed so that the failed plan resumes its execution without any redundancy. We demonstrate such a failure recovery method using an fictional example composite service. Aspect of key features of our system are demonstrated through the web interface.

Being a prototype system, *ASKIT* has demonstrated the feasibility of implementing composite services using team plans. Moreover, the prototype system is developed on Agentcities network and has live interaction with other agent services. Currently, we are working on simple hierarchical team structures, where the team coordination is heavily imposed upon the personal user agent (the abstract team). Future work will look into other team architectures, taking into account joint intention and commitment using belief propagation among team members. We will also look at the Web services in the commercial world to compare the agent and teamwork approach with the standard industry approach.

## 8 Acknowledgments

The project is funded by Australian Research Council (ARC) Linkage Grant LP0218928 (2002-2004) with Agent Oriented Software Pty. Ltd. A shorter version of the paper is presented at Autonomous Agents and Multi-Agent Systems 2003's workshop - the Agentcities: Challenges in open agent environments.

## References

1. AOS. JACK Teams User Guide. Technical report, Agent Oriented Software Pty. Ltd., 2002.
2. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 2001.
3. M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.
4. P. Busetta, R. Rönquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents - components for intelligent agents in Java. Technical report, Agent Oriented Software Pty. Ltd., 2001.
5. F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. Adaptive and dynamic service composition in eFlow. Technical report, Software Technology Lab, Hewlett-Packard Laboratories, 2000.
6. P. Cohen and H. Levesque. Teamwork. Technical Note 504, AI Center, SRI International, Menlo Park, CA, March 1991.
7. M. Georgeff, B. Pell, M. Pollack, M. Tambe, and M. Wooldridge. The Belief-Desire-Intention model of agency. In *Proceedings of Agents, Theories, Architectures and Languages (ATAL)*, 1999.
8. B. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–357, 1996.
9. M. J. Huber. JAM: a BDI-theoretic mobile agent architecture. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 236–243, Seattle, USA, May 1999.
10. G. Kaminka and M. Tambe. Robust agent teams via socially attentive monitoring. *Journal of Artificial Intelligence Research (JAIR)*, 2000.
11. D. Kinny, M. Ljungberg, A. Rao, E. Sonenberg, G. Tidhar, and E. Werner. Planned team activity. In C. Castelfranchi and E. Werner, editors, *Artificial Social Systems*, volume 830 of *Lecture Notes in Computer Science*, pages 227–256. Springer Verlag, 1994.
12. S. McIlraith, T. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems (Special Issue on the Semantic Web)*, 2001.
13. S. Ponnekanti and A. Fox. SWORD: A developer toolkit for web service composition. In *Proceedings of The Eleventh World Wide Web Conference*, 2002.
14. A. Rao and M. Georgeff. An abstract architecture for rational agents. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 1991.
15. Katia P. Sycara, Matthias Klusch, Seth Widoff, and Jianguo Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Record*, 28(1):47–53, 1999.
16. M. Tambe and W. Zhang. Towards flexible teamwork in persistent teams: extended report. *Journal of Autonomous Agents and Multi-agent Systems, special issue on "Best of (ICMAS) 98"*, 2000.