Available online at www.sciencedirect.com
ScienceDirect
Journal of Systems Architecture 52 (2006) 640–653
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

# Ontology-based multi-site software development methodology and tools

P. Wongthongtham [a], E. Chang [a,*], T.S. Dillon [b], I. Sommerville [c]

[a] School of Information Systems, Curtin University of Technology, GPO Box U1987, Perth, WA 6845, Australia
[b] Faculty of IT University of Technology, Sydney, Australia
[c] Department of Computer Science, Lancaster University, Lancaster, UK

Available online 1 September 2006

**Abstract**

The disadvantages associated with remote communication rather than face-to-face communication is a key problem in the multi-site distributed software development environment. Awareness of what work has been done, what task has been misunderstood, what problems have been raised, what issues have been clarified, and understanding of why a team or a software engineer does not follow the project plan, and how to carry out a discussion over a multi-site distributed environment and to make a just-in-time decision are the challenge. Different teams might not be aware of what tasks are being carried out by others, potentially leading to problems such as two groups overlapping in some work or other work not being performed due to misinterpretation of the task. Wrong tasks may be carried out due to ignorance of who to contact to get the proper details. If everyone working on a certain project is located in the same area, then situational awareness is relatively straightforward but the overheads in communications to get together to discuss the problems, to raise issues, to make decisions and to find answers in a multi-site distributed environment can become very large. Consequently, these problems cause project delay and anxiety among teams and managers. Ontologies coupled with a multi-agents system allow greater ease of communication by aggregating the agreed knowledge about the project, the domain knowledge, the concepts of software engineering into a shared information resource platform and allow them to be shared among the distributed teams across the sites and enable the intelligent agents to use the ontology to carry out initial communication and classification with developers when the problem is raised in the first instance. In this paper, we present the key challenges in multi-site software engineering and the ontology representation of commonly shared conceptualisations in software development. We demonstrate the agent communication with developers in the form of man–machine interactions and the great potential of such a system to be used in the future for software engineering in multi-site environments.


*Keywords:* Ontology development; Software engineering ontology; Agent-based system; Multi-site software development

## 1. Introduction

The centralised and single-site software development approach has been widely used by large and medium software development teams. Traditional software development has occurred in environments

* Corresponding author. Tel.: +61 0 8 9266 1235; fax: +61 0 8 9266 7548.
*E-mail addresses:* WongthongthamP@cbs.curtin.edu.au (P. Wongthongtham), ChangE@cbs.curtin.edu.au (E. Chang), tharam@it.uts.edu.au (T.S. Dillon), is@comp.lancs.ac.uk (I. Sommerville).

1383-7621/$ - see front matter 
doi:10.1016/j.sysarc.2006.06.008

where all necessary software development documents and source code reside on a local server available to the developer over the Local Area Network (LAN). However, in today's global economy, collaborative software development, spanning multiple teams in multiple development locations, is becoming the norm rather than the exception [1]. We often see that the teams of developers working on today's software development span many cities, regions and sometimes across several continents. At present it is common for even small to medium sized software development projects to consist of two or more clusters of developers working across several distributed locations. Elder, Audet and Amboise [2,3] state that current software process models and methodologies do not address the issues of collaborative Multi-site Software Development (MSSD). To successfully deliver a large complex information system and to reduce the cost of software development, we often outsource in the sense that we use all global resources presently available to be able to obtain the specialised skills needed. This means that specialised groups will have to work together from remote sites to achieve common integrated development goals.

The following examples illustrate the need for a MSSD methodology:

- In outsourcing situations the analysts who produce the problem definition, system specification or IT solution may be physically far removed from the design and programming teams on either a regional or international basis. This physical distance could become a crucial issue if the specifications are not complete or ambiguous or continually evolving.
- If each implementation group resides at different sites, different interpretation of component specifications or software requirements can lead to incompatible components or sub-systems. This will cause severe integration problems.
- Another example is that if a component-based approach is adopted; components may be developed by multiple remote teams. Although working on the same software, different expert groups have different terminologies and they often find this hard to overcome without face-to-face communication.
- A further example is that customers and testers may be at different sites from the programmer.

Most of the existing software process models assume a centralised approach to software development. As shown in the above examples there are many situations where such an assumption is inappropriate. This assumption is also present in many of the current software engineering methodologies which do not address multi-site software development. Most project management approaches do not consider multi-site software development management either. Lastly, we see that many technologies have not yet become mature enough to facilitate multi-site development [4]. Those process models, methodologies, technologies and approaches cannot be followed linearly for a multi-site situation.

In the remainder of the paper we firstly state the issues in the multi-site software development environment (Section 2) and in the existing global software development approaches (Section 3). Our methodology, utilising ontology-based multi-site software development, is given in Section 4. The paper progresses on to the software engineering ontology in Section 5 which is followed by its implementation in Section 6. Section 7 presents multi-agent based informatics engineering and Section 8 discusses engineering of multi-agent based systems. Section 9 discusses the deployment of the ontology-based multi-agent system. The paper ends with our conclusions in Section 10.

## 2. Issues in multi-site software development (MSSD) environment

We define MSSD as that which involves a software development project that is carried out by multi-teams over multi-sites. With the advent of the Internet, we often see that the system specification is done in one city and the design or implementation or testing is done in another city or cities. We classify five major issues in multi-site software development, namely

1. different concepts and terminologies used in different teams about the principles of software engineering and project management,
2. different level of understanding about the problem domain they are dealing with,
3. different training, different levels of knowledge and skills that exist among teams,
4. issues raised and that cannot be solved in time, and
5. no sense of ownership about the project and the implication that it is always somebody else's fault; time is wasted on arguments rather than on the project.

With advent of the Internet, the management tasks get more complex for these Internet-based multi-site projects and their intrusion into the software development paradigm. This emphasises the need for a new software development paradigm and continuing research in the field of software engineering and project management.

## 3. Issues in existing global software development approaches

In this section we give a brief overview of current global software development approaches. The survey has been focused on the recent initial proposals on multi-site approaches for which our work will be based on.

Computer-Supported Cooperative Work (CSCW) [5] is a well known approach to support group design work in the manufacturing environment. It focuses on person-to-person interaction and communication between the teams. We find that this approach has not been applied to the software development situation and has not addressed group-to-group interaction and communication. It is noted that software production is much more complex than manufacturing problems because software development is a very dynamic process. There are constant changes in requirement, design, and implementation and it is frequently seen that new situations, new terminologies and new concepts appear during the software development period. Software development needs to constantly deal with changes and needs to carry out person-to-person as well as group-to-group interaction. The extension of CSCW by adding the semantics of group-to-group discussion, negotiation and collaboration, and semantic based CSCW will help and support the multi-site multi-team environment for the software development methodology.

Carmel and Agarwal [6] examine how distance contributes to the complexity within organisational processes. Coordination, control, and communication are considered as crucial challenges of distance. They describe three tactical approaches that can be applied across a range of geographical projects:

- Tactic 1: Reduce intensive collaboration. Two or more development units are working together on the same project. This is found to not be the case often in the multi-site software development environments because software tasks are distributed to multi-teams and each team has sole responsibility for its own tasks; so it is often unlikely a task is carried out by two teams in different locations. This would create even more communication traffic between multi-site teams.
- Tactic 2: Reduce cultural distance. For example, American firms generally situate development units in foreign locations where cultural distance is less; such as Ireland, or where language barriers are minimal such as in India or the Philippines. However, we need to use all global resources available to be able to acquire the specialised skills, not only from countries like Ireland or India.
- Tactic 3: Reduce temporal distance. This prescribes using synchronous communication over distance such as videoconferencing and conversations over the telephone. However, it has been found that videoconferencing is expensive for small and medium enterprises and telephone conversations usually cause misunderstanding because one developer may not understand or cannot see a particular error on the screen when the other party talks about something wrong in the user interface on the screen.

Overall this methodology addresses miscommunication and concerns but still has not fundamentally solved the communication problems, even though tactics 2 and 3 are utilised. However, we can see the budget may not be high, especially for the third party small and medium enterprises' (SME) regional companies and remote town teams.

Mockus and Weiss [7] propose a technique of using code change history to compute the degree of relatedness of the work item at two sites. Work could be transferred from a primary site with resource shortages to a secondary site that has under-utilised development resources. They use an algorithm to find the best possible work transfer. The following approaches are being considered in this methodology:

- Transfer by functionality, in which the ownership of a subsystem or a set of subsystems is transferred. A drawback is that if a new functionality is added, it might require using more experts from several sites, thereby increasing the need to coordinate feature work between sites.
- Transfer by localisation, in which developers modify the software product locally for a local market. A drawback is that this

approach requires maintainability experts in all the domains that might require change when adapting the system to the local market.

- Transfer by development stage, in which developers perform different activities at different locations. A disadvantage is the need to communicate and coordinate between sites to proceed to the next development stage.
- Transfer by maintenance stage, in which developers transfer older releases, primarily for the maintenance phase, when they no longer expect to add new features to the release. A disadvantage is a potential decrease in quality and increase in problem resolution intervals because the maintenance stage has not been at the same site as the design and implementation stage. Communication needs between the original site and the maintenance site might increase when difficult maintenance problems require the original site's expertise.

The methodology still does not address the issue of communication between groups over multi-sites.

Repenning et al. [8] proposed using component architecture to support work distributed across sites. They developed the Component-Oriented Rapid Development (CORD) process that is similar to XP. There are two phases in the CORD process:

- Phase 1: Centralised analysis and design: A large group of users, domain experts, designers, and developers analyse project requirements and create application mock-ups and interoperability specifications.
- Phase 2: Distributed analysis, design, implementation and testing: Development is distributed by the group to independent teams.

On the negative side, no detail is given of distributed team work different from the centralised team.

Heeks et al. [9] shows three interesting case studies and captures their successful outsourcing strategies to maximise team value. They consider relationships between central office and outsourcing teams or developers and clients or customers. When successful relationships occur between these parties they call it Synching. In turn, when unsuccessful relationships occur they call it Sinking. They define cocpit dimensions to minimise gaps between the central office, outsourcing teams or developers and clients or customers: coordination/control, objectives and values, capabilities, processes, information, and technology. They show that the three following issues influenced the above six cocpit dimensions: tacit knowledge, informal information and culture. However, we find the description about the methodology and approach are very abstract and at a high level. There is not enough detail to describe the processes, tasks or actions so that the software teams can follow and execute these details, strategies and approaches. We also note that this approach may create pressures and tensions between the clients or customers and the developers which might be acute or chronic [9] rather than improve the development process.

## 4. The ontology-based MSSD methodology

An ontology provides an important mechanism to facilitate producing high-quality software under multi-site environments. Since an ontology has been used to express formally a shared understanding of information [10], it enables the sharing of an agreement among teams distributed across the sites by making assumptions explicit. The key idea is to have agreement explicitly interpreted by software tools rather than just being implicitly interpreted by human developers. The representation of software engineering concepts, software development tasks, software development models, software development processes as well as software development documentation using an ontology and sub-ontology, will provide intuitive, clear, precise concepts and ideas, knowledge and classified issues.

Ontologies are utilised as part of a communication framework for multi-site distributed software development environments. Software engineering concepts, ideas and knowledge, software development methodologies, tools and techniques are organised into a software engineering ontology and used as the basis for classifying the concepts in communication thereby enabling questions, problem solving and sharing solution development and knowledge to be shared between multi-site teams. We propose the use of ontologies in conjunction with intelligent software agents to support multi-site software development. A methodology framework for ontology-based MSSD is illustrated in Fig. 1. As the figure indicates, the system has a software development resource that is for software agents to consult or refer to.

The software development resource consists of

- a generic software engineering ontology,
- a specific software engineering ontology, and
- a knowledge base repository.

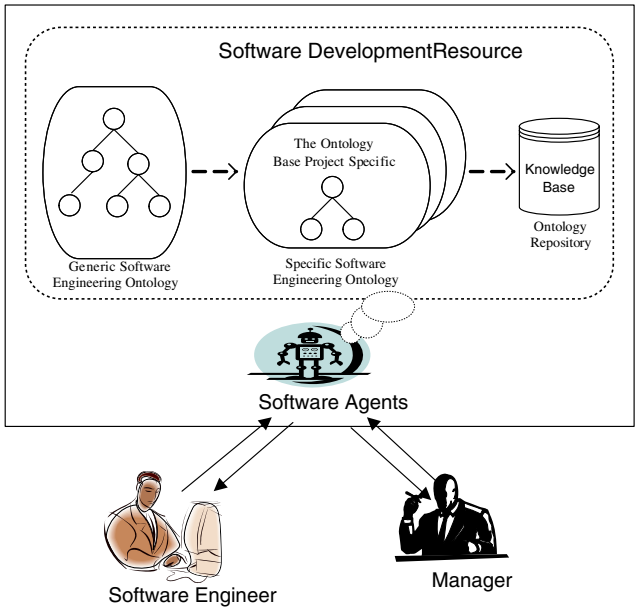


Fig. 1. Methodology framework for ontology-based MSSD.

*Generic software engineering ontology* represents knowledge in the software engineering discipline which is concerned with all processes of software production from the stages of software requirements through software verification and validation. The software engineering discipline covers many aspects of software development. Since each project is different from another, they may only need a subset of the software engineering ontology. Therefore, this information resource allows one to generate a subset of ontological knowledge about software engineering. *Specific software engineering ontology* represents the project specific knowledge and that specifically meets a particular project need. The ontology is to be used for a particular project. Note that the specific software engineering ontology can also be tailored and customised, and once it is created, it should be available to share among the teams. All team members are encouraged to obtain knowledge from it though software agents and studying, and obtaining answers, classifying knowledge and using it as the basis of any conceptual discussion and questions arising, including specification, design and implementation, as well as documentation. *The knowledge base* stores the ontologies along with actual data or project data in an ontology repository. Set in the multi-site distributed software development environment, these resources can be used for sharing of intra and inter project communications of both common knowledge in the software engineering area and project knowledge that have been established by consensus agreement by different project members in dispersed locations. In particular, the specific software engineering ontology fosters a seamless and virtual intra project environment for internal browsing, searching, sharing, and authoring ontological project data across sites.

A software agent is a small computer program that is autonomously *capable* of getting answers from user queries, making decisions based on appropriateness, communicating with other agents and conveying results to the system or the users. Software agents are intelligent, autonomous problem solvers. They have their own goals, capabilities and beliefs what allow them to act intelligently within their field of expertise.

Ontologies coupled with a multi-agents system allow greater ease of communication by aggregating the agreed knowledge about the project and the domain knowledge of software engineering into a shared information resource platform and allow them to be shared among the *distributed* teams across the sites and enable the intelligent agents to use the ontology to carry out initial communication with developers when the problem is raised in the first instance.

The system utilises software agents based computing in the sense that *the* agent has knowledge through consultation with ontologies in the ontology repository. Due to agent capacities in reading and reasoning published knowledge with guidance of the ontology, the shared ontology enables agents to have meaningful communications. We design a set of agents cooperating with each other and interacting with users or team members, and these are

- user agents which represent each team member being provided with services,
- safeguard agent which represents system authentication for user authorisation and access level,
- ontology agent which represents manipulation and maintenance of the software engineering ontology, and
- decision maker agent which represents decision making on the matter of updating the software engineering ontology.

At a remote site, a software engineer having and working with their own repository of software components, documents and codes, etc. interacts with his/her user agent in the system when he/she wants to enquire, to discuss a problem, to raise an issue,

to make a decision, or to find answers in a multi-site distributed environment. If he/she requests to change or update project data and it is beyond his/her user agent to decide then the user agent will communicate with the decision maker agent. The decision maker agent then gathers information from the other team members as well as consults the ontologies from the ontology repository. Making the decision is based on the information obtained from consulting ontologies. The final solution(s) will then be raised up and sent back to the involved software engineers. In case the decision maker agent has difficulties coming up with any solutions, the agent will put it through to the authorised person(s) or team leader to make a decision. Once the agent gets the solution(s) from the person or the team leader, it will automatically reconfigure or update the ontology, the knowledge base in the resources as well as sending back the solution(s) to the involved software engineers. Ontology-based contents and agent-capability descriptions are machine-processable and thus the ontology can be correctly reconfigured by the agents.

## 5. Software engineering ontology

We have merged Gruber's [11], Borst's [12], and Studer's [13] definitions of an ontology as the basis to define the software engineering ontology. Hence, the software engineering ontology is a formal,explicit specification of a shared conceptualisation in the domain of software engineering. Formal implies that the software engineering ontology should be machine-understandable. Explicit implies that the type of software engineering concepts used, and their constraints used are explicitly defined. Shared shows that the ontology specifies consensual knowledge of software engineering which means it is public and accepted by the group of software engineers. Conceptualisation implies an abstract model of having identified the involved software engineering concepts.

An abstract view of creation of the software engineering ontology is shown in Fig. 2 illustrating a transformation to the software engineering ontology. The whole set of software engineering concepts are transformed into the generic software engineering ontology as domain knowledge in the area of software engineering. A particular project or a particular software development probably uses only part of the whole set of software engineering concepts. For example if a project uses object-oriented methodology only, then the concept of data flow diagram will not be included instead it includes concepts like class diagram, activity diagram and so on. The specific software engineering concepts used for the particular software development project are transformed into the specific software engineering ontology as sub domain knowledge. Thus the software engineering ontology is divided into two sub-ontologies which are the generic ontology and the specific ontology. The generic software engineering ontology represents all software engineering concepts while specific software engineering ontology represents some concepts of software engineering for the particular project need. Then in each project there will be project data or actual data or simply some kind of project agreement. The project data specially meet a particular project need and is needed with the software engineering ontology to define instance knowledge. Note that the instance knowledge varies depending on its use for a particular project. Once all domain knowledge, sub domain knowledge and instance knowledge are



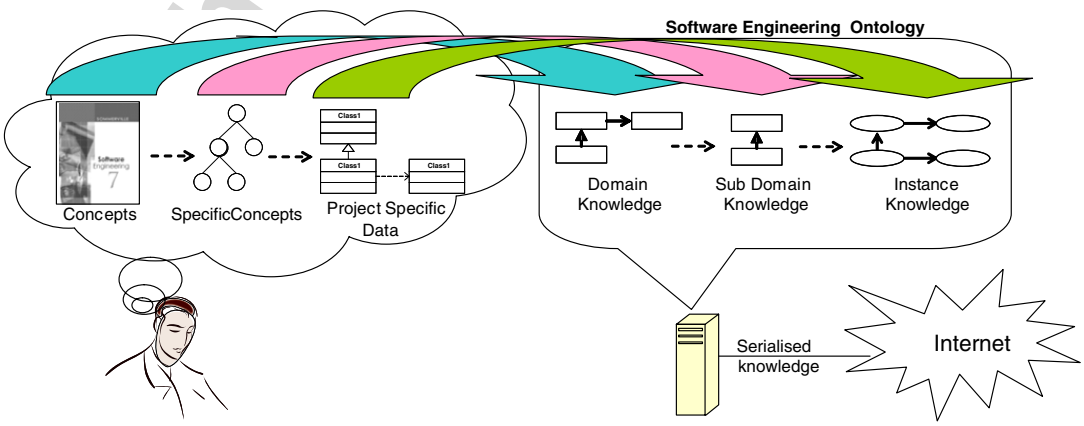


Fig. 2. Schematic overview of software engineering ontology.

created it is available to be shared among software engineers through the internet. All team members, regardless of where they are, can query the semantic linked project data and use them as the common communication and knowledge basis of raising discussion matters, questions, analysing problems, proposing revisions or designing solutions, etc.

## 6. Implementation of software engineering ontology

In this section we present implementation of our software engineering ontology. We will use UML–OCL to represent the ontology and communication architecture. The ontology will be transferred to a software development resource using the web ontology language, OWL, and can be accessed by multi-site, multi-team and multi-development groups. Note that this use of UML–OCL to model the underlying ontology should be distinguished from its use in software development to model the application domain models.

The development of a software engineering ontology basically consists of two processes. Firstly it is the process of creating concepts or ontology classes and relationships that hold among them. The process of defining constraints of the relationships or ontology restrictions is next.

A software engineering ontology class is a concrete representation of software engineering concepts interpreted as sets that contain specific project data. A software engineering ontology instance represents actual data or specific project data. A software engineering ontology property is a binary relation on software engineering ontology classes. Properties are used to create constraints or restrictions that are used to restrict the instances that belong to a class.

For illustration in the paper we show defining concept of an 'Activity' in activity diagram in software design – software engineering domain as an example. Fig. 3 shows a Meta-meta model of an activity diagram representing the activity concept in software engineering domain knowledge. We define class *Activity* as a collection of project data with common characteristics that satisfy a restriction expression of the class. Class *Activity* has its name (*Activity_Name* which is a single string), has a relationship with the class *Swimlane* (*In_Swimlane* which is a multiple object) and has a relationship with itself – class *Activity*. As you can see from Fig. 3, in activity (itself) relationship there is an associated class named *Transition*. It means transitions in the activity diagram i.e., activity transition, branch transition, concurrent transition, and special transition. **Activity transition**, refers to class *Activity Transition* in



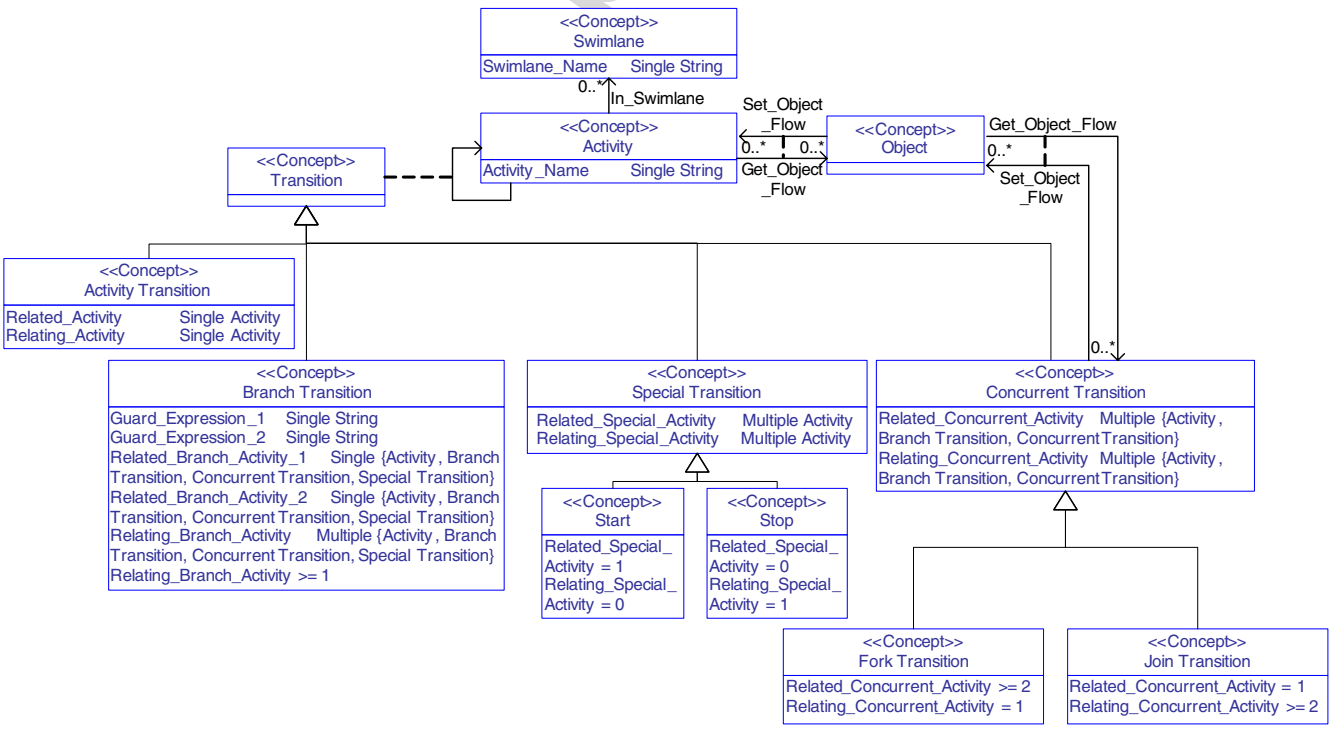


Fig. 3. A meta-model of an activity diagram.

the meta-model in Fig. 3, is where the state of activity transits to another state of activity. We call transiting activity as relating activity while calling transited activity as related activity. The constraints in the transition are that there is one relating activity and one related activity. **Branch transition**, refers to class *Branch Transition* in the meta-model in Fig. 3, is where a decision needs to be made. The constraints in the transition are that there is at least one relating activity named, two related activities and two guard expressions. As you can see from the meta-model in the Fig. 3 a branch transition can link between activities, branch transitions, concurrent transitions, and special transitions. **Concurrent transition**, refers to the class *Concurrent Transition* in the meta-model in Fig. 3, is where some activities occur simultaneously or in parallel. As can be seen from the meta-model in Fig. 3, concurrent transition can link between activities, branch transitions, and concurrent transitions. Concurrent transition consists of join transition and fork transition. The constraints in the join transition are that there are at least two relating activities which join into one related activity. In contrast the constraints in the fork transition are that there is one relating activity split into at least two related activities. **Special transition**, refers to class *Special Transition* in the meta-model in Fig. 3, is where it shows the start and the end of the activity diagram. It consists of start transition and stop transition. The constraints in the start transition are that there is only one related activity and no relating activity. Whereas the constraints in the stop transition are that there is only one relating activity and no related activity.

## 7. Multi-agent based informatics engineering

In this paper we explore the use of software agents for informatics engineering in the software engineering ontology. Team members in the software engineering projects have a natural interaction with each other and share lots of project data/agreement amongst themselves. Since they are not always residing at the same place and face-to-face meetings hardly happen, there is a need for a tool that facilitates effective communication. Traditional software technologies have limitations in coping with distribution and interoperability. Thereby we use agent-based technologies as they were developed to cope with distribution and interoperability [14]. Agents are interoperating and coordinating with each other in peer-to-peer interaction.

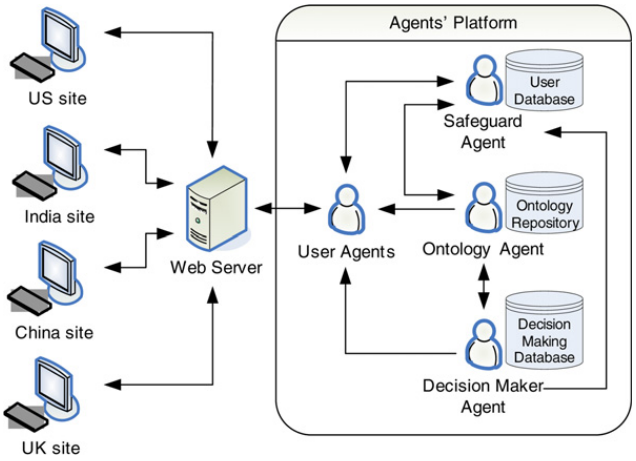


Fig. 4. Model of agents' platform.

The agent based system architecture consists of a set of agents in the multi-site environment shown in Fig. 4. Team members, regardless of where they are, connect to the server via a web browser. This will enable team members to directly use the system without having to download any software or install any application. Each team member is assigned to an individual user agent serving as the communication media for each team member. It uniquely identifies each team member on the system. This allows direct communication between different team members using a messaging system and allows monitoring of the team members' activities. The user agents represent each team member being provided with services, e.g., to query the software engineering ontology. The user agents communicate with the safeguard agent that represents system authentication for user authorisation and determination of the access level. The safeguard agent communicates with the ontology agent if the user agents want to query or update the software engineering ontology. The ontology agent manipulates and maintains the software engineering ontology repository. The ontology agent contacts the decision maker agent if an operation needs to be certified. The decision maker agent is responsible for decision making on the matter of updating the software engineering ontology including acknowledgement of the decision made to all involved team members.

As you can see from the model of the agents' platform only the safeguard agent has connection with the user database. This means that the safeguard agent records all user activities as well. All agents call the safeguard agent and pass information to log all the events that the agents carried out. Thus, tracking can be accomplished by the safeguard agent if

needed. Not only allowing tracing, the safeguard agent can determine bottlenecks if there is any occurrence with the use of the timestamp. The ontology agent is the only one manipulating the software engineering ontology. Thus it is the only one that has access to the ontology repository. All agents contact the ontology agent in the cases of wanting to view, query, or update the ontology. The decision maker agent has its own database to store data for decision making occurring in the system.

## 8. Engineering of multi-agent based system

Jade [15] is used as the platform for building intelligent agents in compliance with the Foundation for Intelligent Physical Agent (FIPA) allowing agents to communicate with the other agents in the same standard. Jade implemented in Java simplifies implementation of multi-agent systems through a middleware. We use the Graphic User Interface (GUI) provided by the platform [16] for monitoring and controlling the status of agents while implementing. For agent communication we use, as provided by JADE, the Agent Communication Language (ACL) [16]. We use communication intentions like inform, propose, query, etc.

The agents themselves not only listen but can decide on a course of action on their own. The agents have some characteristics [17] that define them which are:

- Reactive – the agents themselves perceive circumstances which are different and various e.g., a user via a graphic user interface or other agents.
- Proactive – the agents themselves can have different types of behaviors, they can either be the initiator by carrying out tasks with a goal in mind or be activated by some external events triggering them to do tasks.
- Social – since the agents are proactive, and there are 2 types of agents, initiator and responder, agents are able to communicate with one another and accomplish their goals respectively.

We explain mapping the aforementioned agent properties into design decisions, using an example. The reactive property requires the agents to be active objects with at least one java thread to pursue goals. For example for the decision maker agent, its goal is to sum up the final solution to update in the ontology. It proactively starts from collecting a number of possible solutions, carrying out negotiations, and pursuing the final decision. Besides the agents are social allowing interactions with other agents to pursue their goals. For example the decision maker agent interacts with user agents to collect team member ideas of updating in order to pursue one of its goals: gathering possible solutions.

We also use the following functions JADE provided [16] to deploy a multi-agent system:

- Yellow page service – which provides a directory service for agents to publish their service as well as searching the directory for the services they want.
- Agent life-cycle management – from the creation of a particular agent to the termination of it.
- Message transport service – formats the message with a certain tag and allows easier filtering of different types of messages and easier searching of messages.
- Security – JADE provides a set of security protocols that can be implemented if a secure JADE platform is required.

Details of the four groups of agents are given next.

### 8.1. User agents

A user agent is assigned to every team member that is logged into the system after authentication and provides different services to different access levels for each team member. All team members are provided with the service to query the software engineering ontology. The list below shows the different access level:

- Querying level – only querying service allowed no modification.
- Add and modifying level – restricted access to add and modify service of the software engineering ontology instances (project data). At this level some operations may be required to be made through the decision making system e.g., request for revision of project design model. Simple updates like status of project or documentation update, this would straight away be updated to the software engineering ontology.
- Full access level – unrestricted access to all services provided.

These access levels are given according to the different status of the team members. We also use the hierarchy of the software engineering sub-ontology

categories to determine the access to the status of team members. For example the sub-ontology 'software design' would require access for the designer team or the sub-ontology 'software requirement' would demand access for the analyst team to access, add, and modify their project data. Nevertheless, the team designer can look up project requirements through the sub-ontology 'software requirement' but no changes allowed, which means they are on the querying level. The team leader will have full access including monitoring team member activities.

All agents' lifecycles reside on the server, and each team member is assigned to a user agent when a login is made. Each user agent is an initiator based on the user actions; the agent will carry out the specific operations accordingly. All the operations have different logic involved, but the structure of creating a user agent is the same. Fig. 5 shows the user agent execution model.

In the agent creation process, an agent object is created when a user login is carried out. In the agent listener process, the agent listens to any events based on the user actions. After a fixed waiting time expires, with no any (other) actions from the team member the agent will terminate itself. The agent will setup the required communication in the service creation process setting based on user operation e.g., setting the service type. Then the agent needs to find the identical service type in the Directory Facilitator (DF). The DF contains all the services published by agents that provide their services. The user agent will look up the DF for any services that match the one it is looking for. If the user agent cannot find the service, it deems that the service is unavailable for now and the operation will be terminated. In the process of ACL message setup the user agent creates a message object and sets up the content to be passed to the other agent. Normally it is a request for service. If the particular operation needs to communicate multiple times, or additional tasks need to be carried out the ACL message will then be setup again in a loop action. Once the goal is achieved the user agent will call logger to log all the activities. The user agent will then kill itself if the team member decides to log off the system or the team member is idle for too long otherwise the agent will go back to the listener process.

This is typical behaviour of goal specific agents, which exhibit one-shot behaviour. In other words, the agent is created for a purpose, and once the purpose is achieved, the agent will be terminated. The life-cycle of the user agent listed above applies to all the operations that the user does. And if the user decides to use another operation, a new user agent will be created for that specific operation.

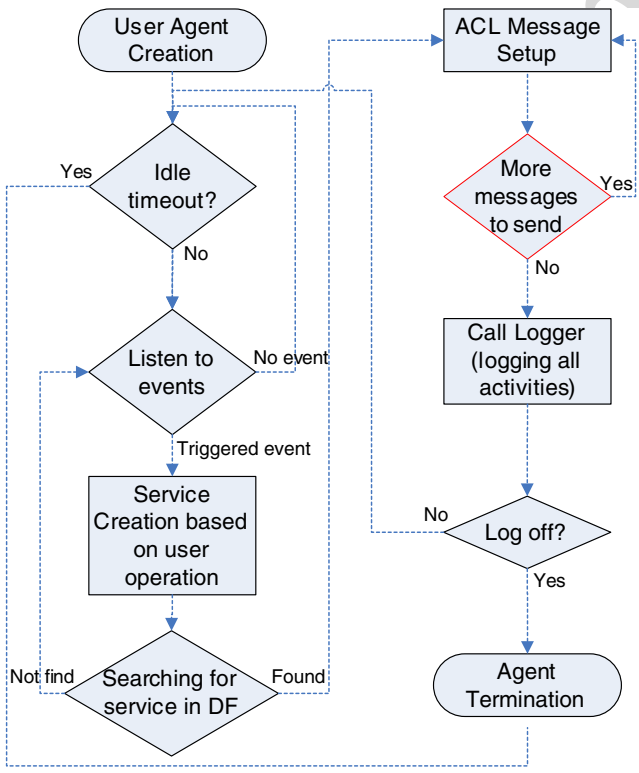


Fig. 5. The user agent execution model.

### 8.2. Safeguard agent

To implement security features into the system, it has been decided to appoint another agent, in this case the safeguard agent. The safeguard agent will be doing user authentication authorisation and allocating access levels. The user identification will be verified with the user database handled by MySQL as well as access level allocation.

There will be a few security levels, mainly:

- Software engineers – they can be either
  - Software engineers, who have no access in updating, and can only do look up.
  - Software engineers who have an access to update project data.
- Team leader/project manager – who have unlimited access to all functions like, updates, backups, ontology maintenance and database access.

It should be noted that only the team leader/project manager has access to the MySQL server. Team

leaders, who do not have any of the access levels stated, will not have the access to that specific section of the ontology.

The safeguard agent is created when the Jade platform is initialised and running. It cannot be terminated by normal means, and only team leader/project manager have access to do Jade platform maintenance and also if due to certain circumstances force shutdown the platform or kill any agent processes. This agent is the only one that connects user agents to the ontology agent; therefore if it is killed or terminated, the system will not be functional at all. Fig. 6 shows the safeguard agents execution model.

In the safeguard agent creation process, the agent is created when the Jade platform is initialised and running. Then the services that are provided by the safeguard agent are initialised, setup and posted to the yellow page, which is the directory of the Jade platform. From this directory, all agents will search and find the service they are looking for. The safeguard agent will idle until another agent establishes a link or makes contact with it. Once there is a connection the service type is checked and if it matches the service type the safeguard agent is providing, further processing will be carried out otherwise the message is discarded and the request is rejected. There will be two possible cases. The first case is that team member requests to view, add, and update the software engineering ontology. In this case the safeguard agent needs to verify the team member account first. If the team member has an authorisation, the safeguard will retrieve the user access level and then pass the request to the ontology agent. The last case is that the safeguard is requested by any agents to record team member activities. Because the safeguard agent is the only agent that connects to the user database thus if there is any action to log into the database, it will be done by the safeguard agent. The safeguard agent will always be alive unless it is killed by human intervention.

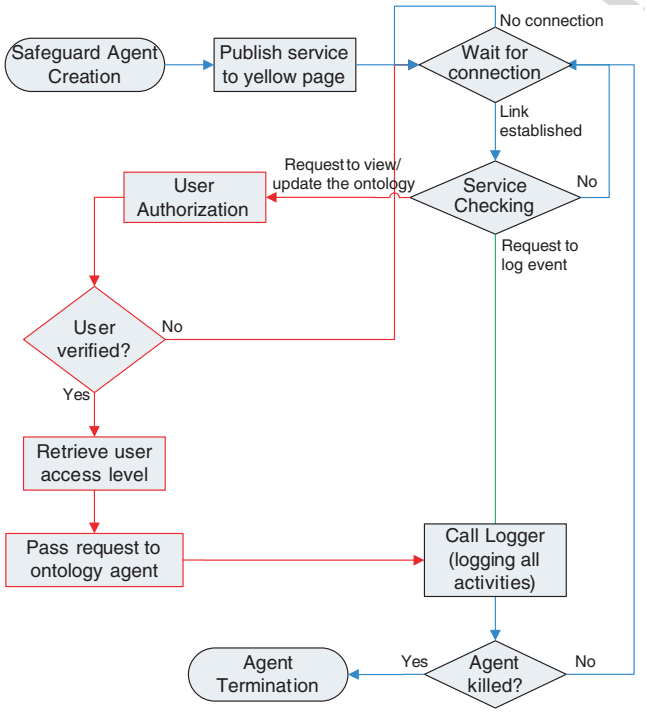


Fig. 6. The safeguard agent execution model.

### 8.3. Ontology agent

The purpose of having an ontology agent is to manage connections with the software engineering ontology.

A team member can search in the generic software ontology for clarification or classification of certain concepts. It also serves as a searching tool to help narrow down the vast amount of concepts in the ontology. Through the use of the ontology search function, the team member can re-classify concepts to match with their project needs. This leads to the specific ontology. Note that the information provided by this function are all in XML format, which means they can be easily manipulated to display only a certain part of the information retrieved or able to provide a different display interface with the same set of information retrieved.

In the specific software engineering ontology which has project data on it, the team member can view, add, and update the project data. The ontology agent will only allow direct updates for the minor changes/updates. An example of a minor change is an enumerated type where the changes allowed are already fixed and team members cannot put in other values. Another example of a minor change is the changing of the status of a document with the option of, for example, 'verified' or 'processing'. By default any updating apart from the minor changes will be done by the decision maker agent and is recorded. Nevertheless there will be another option for a team member to select whether these changes will go through the decision making system or not. In the decision making system the changes will not be updated to the specific ontology straightaway. They need to be verified by all members of the community and therefore need to be stored in the decision making database. After all

members have agreed with the changes, the update will then proceed by the decision making system. The decision making system is handled by the decision maker agent whose details are given in the next section.

The ontology agent acts as a responder by publishing it services to the yellow page and allowing other agents who are looking for the service it is publishing to communicate with it. As the ontology agent is a service provider, it does not terminate itself after finishing a communication with another agent. It will always be executed again whenever an agent communicates with it. This type of behavior is classified as cyclic under Jade and the agent will only terminate if the Jade platform is terminated or forced to be terminated through the Jade GUI. Fig. 7 shows the ontology agent execution model.

In the ontology agent creation process, the ontology agent is created when the Jade platform is initialised and running. Then the services that are provided by the ontology agent are initialised, setup and posted to the yellow page, which is the directory of the Jade platform. From this directory, all agents will search and find the service they are looking for. The ontology agent will be idle until another agent establishes a link or make contact with it. Once there is a connection the service type is checked and if it matches the service type the ontology agent is providing, further processing will be carried out but if not the message is discarded and the request is rejected. There will be two different services i.e., viewing and updating services. For the viewing services, the ontology agent simply carries out querying. Another service, the updating service, the ontology agent considers whether it is a minor change or major change. For a minor change the

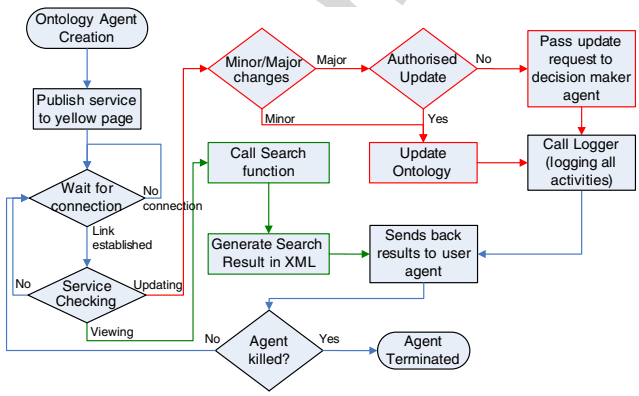


Fig. 7. The ontology agent execution model.

ontology agent updates the ontology straightaway and also records the changes. For a major change, the ontology agent checks whether the update request had been authorised. Basically for major changes, the ontology agent will pass the request of changes to the decision maker agent to precede further processes, for example, gathering information and consulting the ontologies in the ontology repository. On passing through the decision maker agent, the updating can be done by ontology agent. Every activity will be recorded by the call logger process. The results of the process are sent to the user agent that made the enquiry. The ontology agent will automatically wait for another connection and will only be terminated if it is forced to be terminated, either by a kill process or a Jade platform collapse.

### 8.4. Decision maker agent

As the name itself states, the job of the decision maker agent is to make decisions on the matters of major updates requested. The major updates must be agreed upon by every member in the team. The decision maker agent is in charge of sending messages to every team member of the major update requests, gathering and storing possible solutions, making decisions by consulting ontologies and from the rule base, and advising all team members of the solution. Every team member in the project can respond to the debate message but the decision maker agent will give more credit to some involved members and the team leader when it comes to determine what action is needed for the update. Once the decision maker agent gets all the responses, the agent will proceed with decision making and finalising the solution. The following example gives an idea of how the agent progresses the processes. A member requests to update the project design which is consider as a major change. The decision maker agent sends this request to all team members. This will brainstorm all the ideas from analysts, programmer, tester, and everyone. The decision maker agent gathers all possible solutions; for example, there are three possibilities: A, B, and C. A is agreed to by two designers, B is agreed by an analyst and a tester, and C is agreed by a team leader and two designers. As the debate is about project design the decision maker agent gives more credits to the designer (say two credits) and team leader as well (say two credits). For all team members who are not involved in design, the decision

maker agent gives less credit (say one credit). Accordingly, solution C gets more credits, six credits whereas solution A and B get four and two credits respectively. The decision maker agent then makes the decision that solution C will be updated in the ontology and acknowledges this to all team members. In the case of getting equal credits in all the possible solutions, the decision maker agent defers to the team leader. Fig. 8 shows the decision maker agent execution model.

In the decision maker agent creation process, the decision maker agent is created when the Jade platform is initialised and running. Then the services that are provided by decision maker agent are initialised, setup and posted to the yellow page, which is the directory of the Jade platform. From this directory, all agents will search and find the service they are looking for. The decision maker agent will idle until another agent establishes a link with it. Once there is a connection the service type is checked and if it matches the service type the decision maker agent is providing, further processing will be done, if not the message is discarded and the request is rejected. Once the service is activated, the decision maker agent broadcasts messages to all team members. After that the agent gathers responses and makes a decision on which possible solution will be finalised and be updated to the ontology. Updating the software engineering ontology is done by the ontology agent by the decision maker agent forwarding the request to the ontology agent. Then

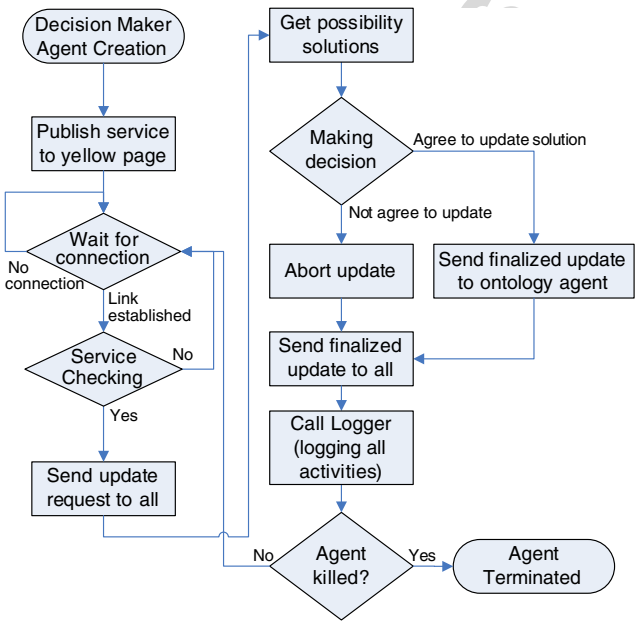


Fig. 8. The decision maker agent execution model.

the decision maker agent sends the final decision message to all team members and records this event. The process of the decision maker agent making a decision has been given in an earlier example. Also the agent can decide to abort the update if there no one agrees upon the update requested. The decision maker agent will automatically wait for another connection and will only terminate if it is forced to be terminated, either by a kill process or a Jade platform collapse.

## 9. Deployment of ontology-based multi-agent system

We have designed and developed man–machine interactions into three platforms i.e., question platform, suggestion platform and solution platform.

### 9.1. Question platform

The question platform stage is basically for raising issue(s) within teams or just querying project data. Team members can use the platform just to query project data to clarify project agreement. If a user has an issue to raise he/she will need to formulate a question elucidated by clarifying the precise project data he/she wants to discuss. Therefore retrieving project data is the main function of this platform. As the user has progressed the querying its diagram is shown and called the original diagram. It could be a UML-like diagram if the debate is on the UML diagrams. The diagrams will help them to have a clear understanding as well as help them recognise the instance knowledge or project data. This is particularly useful when team members work on many projects simultaneously.

### 9.2. Suggestion platform

The suggestion platform is fundamentally for all team members to propose potential solutions of the discussion. Thus this platform mainly involves modifying project data or instance knowledge. In the platform all proposed solutions are pending. After a certain time of gathering some feedback from team members the system (by decision maker agent) determines what action is needed or which solutions will be updated. The updating process is made through the solution platform. After the user has progressed the changes through the platform its diagram is shown and called the suggested diagram. It could be a UML-like diagram if it is about UML

diagrams. The diagrams will help them to have a clear understanding on what they have proposed for the solution.

### 9.3. Solution platform

The solution platform is simply used to reveal a final solution for the discussion. The solution gets updated in the ontology repository at this stage. As the same level with the other two platforms, it shows the solution as a diagram called the solution diagram. It could be a UML-like diagram if it is about UML diagrams.

## 10. Conclusion and future work

We have described how software engineering ontology models provide an approach to transforming explicit semantic knowledge to conceptual knowledge representations and formalise consensus agreement between project team players to approve knowledge as the common communication language and project knowledge across all sites. Also we have shown work on utilising multiple software agents to access and manipulate data from ontologies in an ontology repository and performing reasoning etc. Simple functions like system authentication and complex functions like making decisions have also been carried out by agents. Future work could aim at identifying ways of how agents aggregate knowledge from other ontology repositories in networked systems to reconfigure a generic ontology, which can automatically grow and establish a larger scale of common communications and domain knowledge without human intervention. This concept can also extend to a generic ontology that can self generate and maintain the knowledge.

## References

[1] R. Capasso, Keeping Geographically Distributed Development Team in Sync. 2000, Rational Software.
[2] J.D. Elder, Multi-Site DIVIMP Development: Some Organizational Suggestion, University of Toronto Institute for Aerospace Studies, 1998.
[3] J. Audet, G. Amboise, The multi-site study: an innovative research methodology, An online journal dedicated to qualitative research and critical inquiry 6 (2) (2001).
[4] E. Chang et al., Analysis and development methodology for corporate-wide IS development, in: SCI2003 Orlando, USA, 2003.
[5] A. Crabtree, T. Rodden, S. Benford, Moving with the Times: IT research and the boundaries of CSCW, Compter Supported Cooperative Work (CSCW) The Journal of Collaborative Computing 14 (3) (2005) 217–251.
[6] E. Carmel, R. Agarwal, Tactical approaches for alleviating distance in global software development, IEEE Software 18 (2) (2001) 22–29.
[7] A. Mockus, D.M. Weiss, Globalization by chunking: a quantitative approach, IEEE Software 18 (2) (2001) 30–37.
[8] A. Repenning et al., Using components for rapid distributed software development, IEEE Software 18 (2) (2001) 38–45.
[9] R. Heeks et al., Synching or sinking: global software outsourcing relationships, IEEE Software 18 (2) (2001) 54–60.
[10] T.R. Gruber, Toward principles for the design of ontologies used for knowledge sharing, in: International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, Padova, Italy, Deventer, The Netherlands, 1993.
[11] T.R. Gruber, A translation approach to portable ontology specification, in: Knowledge Acquisition, 1993.
[12] W. Borst, Construction of Engineering Ontologies, Centre of Telematica and Information Technology, University of Tweenty, Enschede, The Netherlands, 1997.
[13] R. Studer, B. VR, D. Fensel, Knowledge Engineering: Principles and Methods, in: IEEE Transactions on Data and Knowledge Engineering, 1998.
[14] M. Wooldridge, Introduction to MultiAgent Systems, first ed., John Wiley & Sons, 2002.
[15] F. Bellifemine, A. Poggi, G. Rimassa, JADE: a FIPA2000 compliant agent development environment, in: The fifth International Conference on Autonomous Agents, ACM Press, Montreal, Quebec, Canada, New York, USA, 2001.
[16] F. Bellifemine, JADE Java Agent DEvelopment Framework, Telecom Italia Lab, Torino, Italy, 2001.
[17] M. Wooldridge, N.R. Jennings, Intelligent agents: theory and practice, The Knowledge Engineering Review 10 (2) (1995) 115–152.