DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

AN AGENT DESIGN AND IMPLEMENTATION USING XML AND RDF TECHNOLOGY

by
Çağlar DURMAZ

May, 2006
İZMİR

AN AGENT DESIGN AND IMPLEMENTATION USING XML AND RDF TECHNOLOGY

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering, Computer Engineering Orientation Program

by
Çağlar DURMAZ

May, 2006
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"AN AGENT DESIGN AND IMPLEMENTATION USING XML AND RDF TECHNOLOGY"** completed by **Çaglar DURMAZ** under supervision of **Prof.Dr. Alp R. KUT** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

.....

Prof. Dr. Alp R. KUT
Supervisor

.....

Jury Member

.....

Jury Member

Prof.Dr. Cahit HELVACI
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to thank my supervisor, Prof. Dr. Alp KUT for his guidance throughout my study. I would also like to thank Tevfik AKTUĞLU for suggesting such a nice subject for my study.

Especially, I would like to express my gratitude to my family for their great supports, encouragements, and helps throughout my study.

Çağlar DURMAZ

AN AGENT DESIGN AND IMPLEMENTATION USING XML AND RDF TECHNOLOGY

ABSTRACT

Firms basically meet their software requirements in two forms. First one is to meet the software requirements as the need of software solution arises. Second one is to meet requirements by getting an overall solution proposing software in one time. First method usually causes a need of integrations among software systems. Second method causes frequent customizations as the needs of firm changes.

This study proposes a solution for integrations and customizations of software systems. It aims to present a solution model for bettering the inter business processes. The model is based on the agent paradigm. As to this model, basically the inter business actors are determined. Agents, representing these actors, are created.

Proposed model does not present an overall solution. The model wraps the present applications in firms. It aims to increase the benefits of the present systems and their rightly usages according to the evolving needs.

In this direction this model is implemented for a firm in production sector. The system getting developed for the firm aims to speed up the production planning processes.

Keywords: agent, multi-agent system, agent based workflow management system, agent based process management system.

AJAN TASARIMI VE XML VE RDF TEKNOLOJİLERİNİ KULLANARAK UYGULANMASI

ÖZ

Firmalar temel olarak yazılım gereksinimlerini iki şekilde karşılarlar. İlki, yazılım ihtiyacı doğduğunda yazılım gereksinimini karşılamaktır. İkincisi, genel çözüm sunan yazılımı bir kere alarak gereksinimleri karşılamaktır. İlk metot, yazılım sistemleri arasında entegrasyon ihtiyacına sebep olur. İkinci metot, firmanın ihtiyaçları değiştikçe sık uyarlamalara sebep olur.

Bu çalışma, yazılım sistemlerinin entegrasyonu ve uyarlamaları için bir çözüm önermektedir. İşletme içi süreçlerin iyileştirilmesi için bir çözüm modeli sunmayı amaçlamaktadır. Bu model ajan paradigmasına dayanmaktadır. Bu modele göre, temel olarak işletme içi aktörler belirlenir. Bu aktörleri temsil eden ajanlar yaratılır.

Önerilen model toplu bir çözüm sunmaz. Model firmalardaki var olan uygulamaları sarar. Model, gelişen ihtiyaçlara göre hali hazırdaki sistemlerin faydalarını ve doğru kullanımlarını arttırmaya çalışır.

Bu doğrultuda model üretim sektöründeki bir firma için uygulanmıştır. Firma için geliştirilen sistem, üretim planlama süreçlerini hızlandırmayı amaçlamaktadır.

Anahtar Sözcükler: ajan, çoklu ajan sistemi, ajan tabanlı iş akış yönetim sistemi, ajan tabanlı süreç yönetim sistemi.

CONTENTS

Page

CHAPTER ONE

INTRODUCTION

Firms meet their software needs in two main forms. First one is purchasing several software programs, which are designed for specific domains, as the need of software arises. Second is purchasing a software solution, which meets almost all of the needs in an ordinary firm, at once.

As to their requirements, a lot of firms buy several software programs in different times. These software programs are usually ongoing their functions separately without being aware of each other. In fact, these software programs are the pieces of one big system. These programs actually provide data input and output to each other, but they are not interconnected. Because software solutions are demanded in different times by different groups, no connection could be found. These connections are provided in several ways by the company workers manually.

Besides, in present days a lot of firms have MRP/ERP system solutions. These systems are tried to settle in the firm at once. This case arises because of the natural necessity of these systems. They have a lot of aims such as resource planning, production and process tracking, purchasing, and marketing. However, accountancy departments get benefited of MRP/ERP systems most. Other departments such as production, purchasing, marketing departments can not get the required return as expected. These systems present not specific but a general solutions for the systems of firms. Therefore, the software must be customized to the firm. The customization efforts are often disappointing. This disappointment happens sometimes because of the software itself, sometimes because of the customization team and sometimes because of both. Even if, the software solutions are implemented into the firm perfectly, some changes will be needed because the firms are in dynamic environments and the needs of them are changing, evolving continuously.

Integration and customization needs will exist forever for the firms. Because remarkable amount of time and money have already been spent for legacy systems (A legacy system is an existing computer system or application program which continues to be used because the organization does not want to replace or redesign it.), legacy systems in firms can not be replaced by other new solutions easily. Instead, new solutions can wrap legacy systems and so new abilities can be added into the system without throwing them into trashcan. Legacy systems are also reliable systems for the workers of the firm. They are reliable because these systems have been tested for a long time. Besides that, workers always like to work in the way they are familiar.

This study proposes a solution for integrations and customizations of software systems. It aims to present a solution model for bettering the inter business systems. The model is based on the agent paradigm. Briefly, this model offers that the inter business actors are determined and agents, presenting these actors, are created.

This study suggests using software agents to meet dynamic software needs of firms. Agents can be defined to be autonomous, problem-solving computational entities capable of effective operation in dynamic and open environments. They have ability of being wrapper around systems. Agents can use the abilities of legacy systems and interact with other agents which use abilities of other domain systems. For example, assume a firm in which a simple MRP software and purchasing software are used separately by production and purchasing departments. To integrate these two systems would be more economic than purchasing a bigger software solution and implementing it. This integration can be realized by wrapping two systems with two agents and letting agents to communicate each other to act like one big system. This ability and other abilities of agents are introduced in chapter two. The new paradigm of agent paradigm, multi-agent systems, and current phase of multi agent systems are also discussed in chapter two.

1.1 Workflow Management and Agents

Workflow management is a promising technology aiming at the automation of business processes to improve the speed and efficiency of an organization. In recent years, workflow management systems (WfMS) have been widely used in business process controlling and monitoring. With the increased complexity, uncertainty and risk in business operations, there is an increased demand on flexible and dynamic workflow management (Chung & his friends, 2003).

A workflow management system (WfMS) is the software that automates the co-ordination and control of tasks during business process execution. The workflow approach helps to separate the business logic represented by business process from the underlying information systems that support the process. This separation allows business processes to be designed without requiring major changes to be made to the underlying computing infrastructure (O'Brien & Wiegand, 1998). The success of workflow paradigm is based on its ability to support modeling, simulation, automated execution, and monitoring of processes in an environment that is distributed, heterogeneous, and only partially automated. While workflow technology has seen an explosion of interest and advances in recent years, numerous technical challenges have been addressed to provide flexible workflow management systems required by complex and dynamic application domains (Chung & his friends, 2003).

If a good system design based on agents is accomplished and every step of a process is taken by agents, this leads the agent based system to be an Agent Based Process Management System (APMS). In this scenario, the whole business process is formed by the pieces of sub-networks within those agents. The process logic is embedded in the agents, rather than being explicitly represented in a centric module. (A traditional WfMS has a centric module that contains the logic of the process.) Thus a central workflow engine is unable to get all the information of the whole business process in order to control it. A more likely solution is to have one

workflow engine residing in each organization or unit. Through interactions among the multiple workflow engines the whole business process is fulfilled.

This study suggests that integrations among application domains should be implemented by agents if automation of processes in firms is desired. This automation won't be designed at one time. This will start with two communicating agents solving a basic problem. The third one is included when another problem is desired to be solved with agents and this will goes on till a whole process is implemented. Any agent can't have a full knowledge of the process but there is a big and modular knowledge about the business processes in this system. The modularity comes from the ability of changing of the way of problem solving inside an agent without any change in other agent domains. So that, the policies and/or goals of firm can be added, discarded and changed easily.

1.2 Implementation Framework for APMS

This study does not only introduce a model for interacting legacy systems and establishing a process management system, it also presents an implementation framework, which is called SADE (Simple Agent Development Environment), for Agent Based Process Management Systems. SADE framework is introduced in chapter five. In this implementation framework, several other frameworks and standards are used. These are FIPA standards, Jade framework, RDF and Jena framework. Chapter three gives an overview on these technologies.

The Foundation for Intelligent Physical Agents (FIPA) standard is a standard for developing and setting computer software standards for heterogeneous and interacting agents and agent-based systems. Jade (Java Agent Development Environment) is a robust and efficient middle-ware for "agent" systems. It complies with the FIPA specifications. Resource Description Framework (RDF) is a good choice for message content language for messages sent among agents. Resource Description Framework (RDF) is a family of specifications for a metadata model that is often implemented as an application of XML. Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for

RDF, RDFS (RDF Schema) and OWL (Web Ontology Language), including a rule-based inference engine. Jena is used in SADE framework. Because Jena is straightforward tool, it won't be mentioned anymore in this study. Anyone can apply to the web address of Jena, (http://jena.sourceforge.net/), for further information.

In chapter six, workflows about production planning are going to be implemented by SADE framework. The system of an organization in sector HVAC (heating, ventilation and air-conditioning) is examined for this study. Fan coil department of the organization uses a simple program working on MS Access. (Fan Coil is an indoor component of a heat pump system, used in place of a furnace, to provide additional heating on cold days when the heat pump does not provide adequate heating.) This simple desktop program is used for production planning. SADE framework will wrap the abilities of this program and connect it with the other legacy systems and departments of the firm. The multi-agent system getting developed for the firm, aims to speed up the production planning processes.

1.3 The Aim of The Study

This study aims to present a solution model for integrations and customizations of software systems and bettering the inter business systems. The model is based on the agent paradigm. Briefly, this model offers that the inter business actors are determined and agents, presenting these actors, are created.

This study does not only introduce a model for interacting legacy systems and establishing a process management system, it also presents an implementation framework, which is called SADE (Simple Agent Development Environment), for Agent Based Process Management Systems. Also an application system about production planning is implemented by using SADE.

CHAPTER TWO

LITERATURE REVIEW

This chapter reviews the major concepts, agent paradigm and multi-agent systems, and related literature about integration of software applications in the subject of workflow management.

2.1 Agents and Multi-Agent Systems

Agent based system is a collection of autonomous computational elements, independent programs (hereafter we will refer to these elements as agents) that perform collective behaviour in order to meet either their individual goals. Agents can be defined to be autonomous, problem-solving computational entities capable of effective operation in dynamic and open environments. Agents are deployed in environments in which they interact, and sometimes cooperate, with other agents (including both people and software) that have possibly conflicting aims. Such environments are known as multi-agent systems.

Agent paradigm is based on the agent abstraction, a software component that is autonomous, proactive and social (Bellifemine, Caire, Poggi, Rimassa, 2003):

- Autonomous: agents have a degree of control on their own actions, they own their thread of control and, under some circumstances, they are also able to take decisions;
- Proactive: agents do not only react in response to external events (i.e. a remote method call) but they also exhibit a goal-directed behaviour and, where appropriate, are able to take initiative;
- Social: agents are able to, and need to, interact with other agents in order to accomplish their task and achieve the complete goal of the system via some kind of agent communication language.

Agents can be distinguished from objects (in the sense of object-oriented software) in that they are autonomous entities capable of exercising choice over their actions and interactions, and may act to achieve individual objectives. They are able to exercise autonomy by choosing how to perform the tasks assigned to them or by deciding on operational tasks to satisfy user objectives. More importantly, they make these choices in the context of dynamic environments in which they are deployed. Agents cannot, therefore, be directly invoked like objects but can be assigned tasks by their owners. Nevertheless, they may be constructed using a wide range of technologies, including object technology, Web Services and others. (Luck, McBurney, Shehory & Willmott , 2004)

2.1.1 The Peer-to-Peer model

A clarification of the proper model, peer-to-peer, for the realization of multi-agent system may be beneficial at this stage.

"Client-Server" (C/S) is the reference model, well-known and widely-diffused, for distributed applications. The model is based on a rigid distinction of roles between the client nodes (resource requester) and the server nodes (resource providers). The server nodes provide the services, more in general the capabilities of the distributed system, but they are not capable of taking any initiative as they are fully reactive and they can just wait for being invocated by the client nodes. Client nodes, as opposite, concentrate all the initiative of the system: they access and use the services, typically, but not necessarily, upon user requests, but they do never provide any capability. Clients can appear and disappear at any time; generally, they have dynamic addresses, while servers must typically provide some guarantees of stability and generally listen to a well-known and static address. (Bellifemine, Caire, Poggi & Rimassa, 2003)

Clients communicate with the servers, but they cannot communicate with other clients. On the other hand, server can not communicate with their clients until the clients have taken the initiative and decided to activate a communication session with the server.

In the peer-to-peer model, in fact, there is no more any distinction of roles and each peer is capable of a mix of initiative and capability: each node can initiate the communication, be subject or object of a request, be proactive, provide capabilities; the application logics is no more concentrated on the server but distributed between all the peers of the network; each node is capable of discover each other, it can enter, join or leave the network anywhere anytime. The system is fully distributed as well as the value of the service is distributed across the network and new business models might be enabled. (Bellifemine, Caire, Poggi & Rimassa, 2003)

An important consequence of the differences between the 2 models is the way the nodes can be discovered. In the C/S systems, clients must know their servers but they do not need to know other clients (of course, given that client-to-client communication is never expected to happen). In P2P systems, who-knows-whom is fully arbitrary and the system must provide proper services that allow peers to enter, join, or leave the network at any time as well as to search and discover other peers. These services are usually the white and yellow page mechanisms that allow publishing and discovering the features and the services offered by a peer. On the basis of the implementation of these mechanisms, two basic P2P network models can be identified (see figure 2.1): pure P2P networks (also called decentralized), and hybrid P2P networks (also called with central index). A pure P2P network is fully decentralized and the peers are fully autonomous. The absence of any reference node makes more difficult to maintain the coherence of the network and the discovery of the peers, with a complexity and bandwidth that tends to grow exponentially with the number of nodes. Also security is quite demanding as each node is entitled to join the network without any control mechanism. The hybrid architectures, instead, are based on a special node that provides a service that simplifies the look-up and discovery of the active peers, their list of capabilities, and their list of provided services. These types of networks, usually, generate less traffic and are more secure as they tend to require so the registration and authentication of the peers. On the other hand, their functioning depends on the availability of the index nodes that might become a central point of failure and attack. (Bellifemine, Caire, Poggi & Rimassa, 2003)

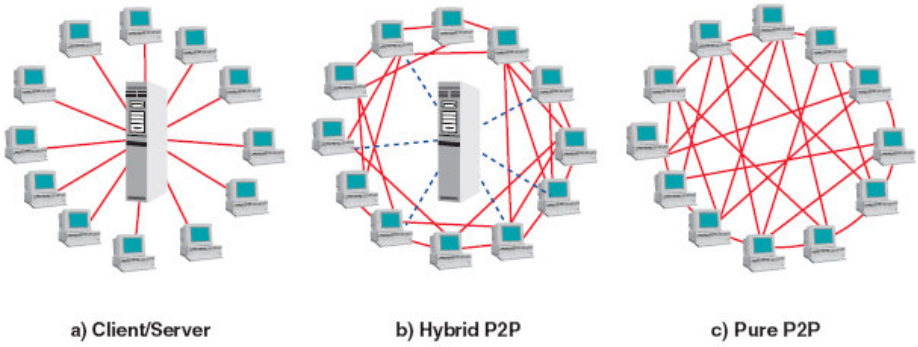

Figure 2.1 Client/Server (left), pure P2P (right), hybrid P2P (centre)

2.1.2 Complex Systems

Modern software and technological systems are among the most complex human artifacts, and are ever-increasing in complexity. Some of these systems, such as the Internet, were not designed but simply grew organically, with no central human control or even understanding. Other systems, such as global mobile satellite communications networks or current PC operating systems, have been designed centrally, but comprise so many interacting components and so many types of interactions that no single person or even team of people could hope to comprehend the detailed system operations. This lack of understanding may explain why such systems are prone to error.

Whether such complex, adaptive systems are explicitly designed or not, their management and control is vitally important to modern societies. Agent technologies provide a way to conceptualize these systems as comprising interacting autonomous entities, each acting, learning or evolving separately in response to interactions in their local environments. Such a conceptualization provides the basis for realistic computer simulations of the operation and behaviour of the systems, and of design of control and intervention processes (Bullock & Cliff, 2004). For systems that are centrally designed, such as electronic markets overlaid on the Internet, agent technologies also provide the basis for the design and implementation of the system itself. Indeed, it has been argued that agent technologies provide a valuable way of

coping with the increasing complexity of modern software systems (Zambonelli & Parunak, 2002), particularly the characteristics of pervasive devices, ambient intelligence, continuous operation (allowing no downtime for upgrades or maintenance), and open systems.

2.1.3 The Agent Paradigm

Agent-based systems technology has generated lots of excitement in recent years because of its promise as a new paradigm for conceptualizing, designing, and implementing software systems. This promise is particularly attractive for creating software that operates in environments that are distributed and open, such as the internet (Sycara, 1998).

Most researchers in AI to date have dealt with developing theories, techniques, and systems to study and understand the behavior and reasoning properties of a single cognitive entity. AI has matured, and it endeavors to attack more complex, realistic, and large-scale problems. Such problems are beyond the capabilities of an individual agent. The capacity of an intelligent agent is limited by its knowledge, its computing resources, and its perspective (Sycara, 1998).

The most powerful tools for handling complexity are modularity and abstraction. Multi-agent systems (MASs) offer modularity. If a problem domain is particularly complex, large, or unpredictable, then the only way it can reasonably be addressed is to develop a number of functionally specific and (nearly) modular components (agents) that are specialized at solving a particular problem aspect. This decomposition allows each agent to use the most appropriate paradigm for solving its particular problem. When interdependent problems arise, the agents in the system must coordinate with one another to ensure that interdependencies are properly managed. Furthermore, real problems involve distributed, open systems. An open system is one in which the structure of the system itself is capable of dynamically changing. The characteristics of such a system are that its components are not nor known in advance; can change over time; and can consist of highly heterogeneous agents implemented by different people, at different times, with different software

tools and techniques. Perhaps the best-known example of a highly open software environment is the internet. The internet can be viewed as a large, distributed information resource, with nodes on the network designed and implemented by different organizations and individuals. In an open environment, information sources, communication links, and agents could appear and disappear unexpectedly. Currently, agents on the internet mostly perform information retrieval and filtering. The next generation of agent technology will perform information gathering in context and sophisticated reasoning in support of user problem solving tasks (Sycara, 1998).

A MAS can be defined as a loosely coupled network of problem solvers that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver (Durfee & Lesser 1989).

The characteristic of MASs are that (1) each agent has incomplete information or capabilities for solving the problem and. thus, has a limited viewpoint; (2) there is no system global control; (3) data are decentralized; and (4) computation is asynchronous (Sycara, 1998). The motivations for the increasing interest in MAS research include the ability of MASs to do the following:

First is to solve problems that are too large for a centralized agent to solve because of resource limitations or the sheer risk of having one centralized system that could be a performance bottleneck or could fail at critical times.

Second is to allow for the interconnection and interoperation of multiple existing legacy systems. To keep pace with changing business needs, legacy systems must periodically be updated. Completely rewriting such software tends to be prohibitively expensive and is often simply impossible. Therefore, in the short to medium term, the only way that such legacy systems can remain useful is to incorporate them into a wider cooperating agent community in which they can be exploited by other pieces of software. Incorporating legacy systems into an agent society can be done, for

example, by building an agent wrapper around the software to enable it to interoperate with other systems (Genesereth & Ketchpel 1994).

Third is to provide solutions to problems that can naturally be regarded as a society of autonomous interacting components agents. For example, in meeting scheduling a scheduling agent that manages the calendar of its user can be regarded as autonomous and interacting with other similar agents that manage calendars of different users (Garrido & Sycara 1996).

Fourth is to provide solutions that efficiently use information sources that are spatially distributed. Examples of such domains include sensor networks (Corkill & Lesser 1983), seismic monitoring (Mason & Johnson 1989), and information gathering from the internet.

Fifth is to provide solutions in situations where expertise is distributed. Examples of such problems include concurrent engineering (Lewis & Sycara 1993), health care, and manufacturing.

Sixth is to enhance performance along the dimensions of (1) computational efficiency because concurrency of computation is exploited (as long as communication is kept minimal, for example, by transmitting high-level information and results rather than low-level data); (2) reliability, that is, graceful recovery of component failures, because agents with redundant capabilities or appropriate inter-agent coordination are found dynamically (for example, taking up responsibilities of agents that fail); (3) extensibility because the number and the capabilities of agents working on a problem can be altered; (4) robustness, the system's ability to tolerate uncertainty because suitable information is exchanged among agents; (5) maintainability because a system composed of multiple components-agents is easier to maintain because of Its modularity; (6) responsiveness because modularity can handle anomalies locally, not propagate them to the whole system; (7) flexibility because agents with different abilities can adaptively organize to solve the current problem; and (8) reuse because functionally specific agents can be reused in different

agent teams to solve different problems. MASs are now research realities and are rapidly having a critical presence in many human-computer environments (Sycara, 1998).

2.1.4 Current Phase of Multi-Agent Systems

Agent paradigm promises a lot of facilities for solving problems of complex, integrated systems. Besides that, ready-made tools may not be considered to be enough for some complex systems. And also designing methodologies for multi-agents are not so mature.

Multi-agent systems are currently typically designed by one design team for one corporate environment, with participating agents sharing common high-level goals in a single domain. These systems may be characterized as closed. The communication languages and interaction protocols are typically defined by the design team prior to any agent interactions. Systems are usually only scalable under controlled, or simulated, conditions. Design approaches, as well as development platforms, tend to be ad hoc, inspired by the agent paradigm rather than using principled methodologies, tools or languages. Although this is still largely true, there is now an increased focus on, for example, taking methodologies out of the laboratory and into development environments, with commercial work being done on establishing industrial-strength development techniques and notations. As part of this effort, some platforms now come with their own protocol libraries and force the use of standardized messages, taking one step towards the short-term agenda.

It remains true that, for the foreseeable future, there will be a substantial commercial demand for closed multi-agent systems, for two reasons. First, there are very many problems that can be solved by multi-agent systems without needing to deal with open systems, and this is where many companies are now realizing business benefit. Second, in problems involving multiple organizations, agreement among stakeholders on the objectives of the open system may not always be readily achieved, and there may also be security concerns that arise from consideration of open systems. While progress on Technologies for open systems will change the

nature of agent systems, the importance of closed, well protected systems must not be underestimated.

2.1.5 Summary of Agent Paradigm

Agent paradigm presents a new view point for system designers. One of the properties of complex, open systems is the existence of plenty actors in them. Every actor has at least a goal and usually more then one goal. These goals usually conflicts with each other. To find a solution in a traditional point of view like designing system via object oriented software, would not represent the real system. If traditional way is chosen, the system would always tend to take actions in several same ways. The environmental changes may not be simulated as good as agent systems. And also to put a new goal into the system is so painful in traditional way. There will be a suspicious about whether if all the aspects may be considered or not. On the contrary, agent paradigm provides modularity. A new goal is placed into system without so much worry. Agent systems force the system designers to define the environment, actors, and goals of actors at the beginning of projects.

2.2 Workflow and Workflow Management Systems

Although many organizations have adapted the Information Technology (IT) to improve their working efficiency, the business processes within their organizations and their partners have not been clearly described and solved. During the execution of business processes, there are not enough techniques and methods to follow-up and control the processes. This leads to the misunderstanding of responsibilities. Workflow management technology tries to overcome these shortcomings. It promises to provide an efficient way to model and control the complex business processes within and between organizations.

A workflow is a composite activity consisting of tasks involving a number of humans, databases, and specialized applications (Hubns & Singh 1998). Workflow refers to group activity automation by task sequencing and information routing (Takeda, Inaba & Sugiara, 1996) .Thus, workflow is a collection of tasks organized

to accomplish some definite business processes. An activity can be performed by one or more software systems, one or a team of human, or a combination of them (Turoff, Hiltz, Bieber, Fjermestad & Rana, 1999). This definition applies the workflow concept to automate business processes. Workflow management involves the (re)design and the (re)implementation of workflows as the needs and the goals of an enterprise.

The production management system used by most of today's manufacturers consists of a set of separate application softwares, each for a different part of the planning, scheduling, and execution processes (Vollmann, 1992).

Workflow is the implementation and automation of a particular business process. A Workflow Management System (WMS) is the software which automates the co-ordination and control of tasks during business process execution (Workflow Management Coalition, 1996).

Different WMS exist to suit different types of business process; these have been classified into administrative, ad hoc, production and collaborative workflow (Sheth, 1995):

- Administrative workflow systems involve repetitive, predictable processes with simple task co-ordination rules. Examples of administrative workflows would be routing documents with in an organization, which involves standardized tasks, tightly linked, and performed regularly.
- Ad hoc workflow systems involve more human co-ordination where both process and information are relatively unstructured. An example would be a sales process which can be relatively unstructured and which becomes structured during the execution of the business process. In a sales process decisions can be made during its execution which will determine subsequent tasks. Ad hoc workflow systems rely heavily on human involvement in controlling and coordinating tasks.

- Production workflow systems handle complex business processes which are more critical to an enterprise. These typically involve some form of transaction processing and require accessing multiple information systems. This would include processes such as customer handling and exception handling processes.
- Collaborative workflow supports business critical processes which are less structured and more suited to collaborative working technologies such as Lotus Notes. This is where the group working technologies overlap with the workflow market.

WMSs have certain limitations that need to be addressed. In commercial environments decisions are not always clear cut but involve the balancing of various vested interests and business policies, and resource levels can change. Such business processes highlight a number of shortcomings in existing workflow management systems (Trammel, 1996). They lack:

- Reactivity: workflow management systems require an a priori representation of a business process and all potential deviations from that process.
- Semantics: many workflow management systems lack an appreciation of the content of a business process and do not make decisions based on the nature of the information generated by a business process.
- Resource management: workflow management systems do not control the resourcing of a business process and so rely on a business process being dimensioned beforehand.
- Heterogeneity: workflow management systems tend to take a centralized view with a single workflow management engine that does not operate across multiple-server platforms or multiple client operating systems.

Yan and his friends have added two lacks of WFMSs like below (Yan, Maamar & Shen, 2001);

- Lack of automation: WFMSs only determine the process logic, but most of the activities are still fulfilled by human. WFMSs can't even start a workflow without human's intervention.
- Lack of generic interfaces: WFMSs need to exchange data between activities or interface to other applications. Currently, these operations depend on API calls. There should be some generic interfaces to eliminate the effort to develop interfaces between WFMSs and other applications.

2.3 Agent-Based Process Management Systems

Integration of workflow and agent technology has recently attracted a lot of attention of researchers in recent years. Agent-based Process Management Systems (APMS) extend the automation of business process management beyond that covered by WMS. The management of a business process can be viewed as consisting of three stages: creation, provisioning and enactment (Jennings, 1996). The creation stage is predominantly a manual activity which involves the analysis, modeling and definition of the business process. The provisioning stage involves the assignment of resource, including people, equipment, computing time, to support a business process. This requires the negotiating, planning and scheduling to ensure that there is sufficient resource to handle expected throughput of work for a given business process. Lastly, the enactment stage involves the management activities required to ensure that each instance of a business process is effectively executed. This includes routing of work, passing of information, activation of automated activities, and the handling of work lists (O'Brien & Wiegand, 1998).

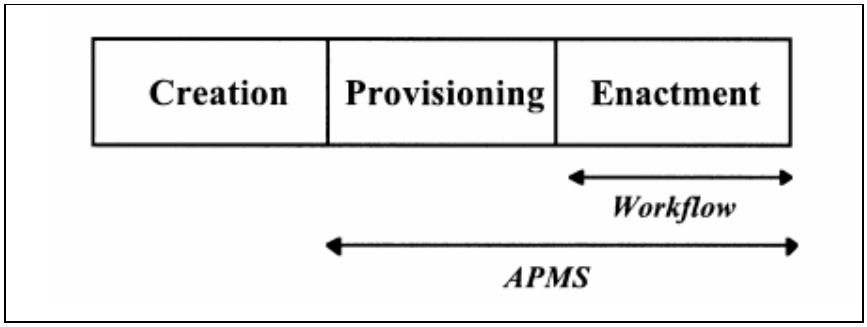

Figure 2.2 Business process lifecycle

Existing workflow technology automates the enactment phase of the life-cycle. APMS extend this into the automation of the provisioning stage of the life-cycle model. Subsequently the dimensioning of business processes is brought on-line and integrated with process enactment, resulting in improved re-deployment of resources and increased flexibility during exception handling. Therefore, unlike WMS which are focused solely on the enactment of process tasks, APMS have two objectives, firstly, the timely execution of business tasks and secondly, the efficient use of resources (O'Brien & Wiegand, 1998).

Applications of agents to workflow management systems can be classified into two forms: agent-enhanced workflow management and agent-based workflow management (Yan, Maamar & Shen, 2001).

2.3.1 Agent-Enhanced Workflow Management

Agent-enhanced workflow management is the basic form for application of agents to workflow management (see figure 2.3). There is one central workflow engine, which controls all the activities. Agents are invocated during the execution of one work item to implement certain tasks. Workflow system controls the generation and elimination of the agents. There are several things agents can achieve in this scenario (Yan, Maamar & Shen, 2001):

- Human interface: it is part of the workplace environment the workflow system provides to its user. The agent acts as a personal assist. Some typical usages are sorting emails, replying email, reminding events, acquiring work items. Example is A1 in figure 2.3.
- Autonomous activity: it implements the tasks autonomously without human's interruption. Example is A2 in figure 2.3.
- Interface to other applications: agent can provide interface to other applications. Instead of defining API for application interoperation, semantic messages can be defined to exchange high-level information. This is also a direction for designing generic interface to applications.

Agents can carry out many tasks without human involvement. From the system view, agent does not necessarily interact with each other. In fact, the workflow engine controls their actions. The information exchanging is through workflow engine. The workflow engine is responsible to create and eliminate the agents. The agents in this scenario do not have to be “intelligent”. In most commercial WFMS products, the agent is more like a piece of ordinary software like this scenario, e.g. IBM WebSphere MQ Workflow (IBM, 2005) and InConcert (InConcert, 2000).

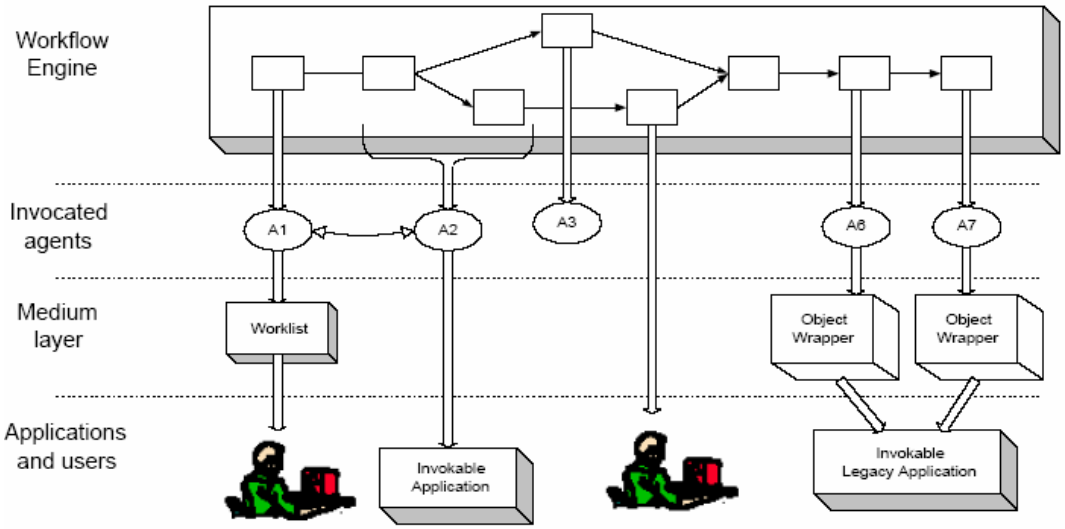

Figure 2.3 Agent Enhanced Workflow Management System

2.3.2 Agent-Based Workflow Management

An Agent-based workflow system is a distributed system consisting of multiple agents (see figure 2.4). These agents are independent to each other and each is responsible for process execution. In this scenario, the whole business process is formed by the pieces of sub-networks within those agents. The process logic is embedded in the agents, rather than being explicitly represented elsewhere (Yan, Maamar & Shen, 2001).

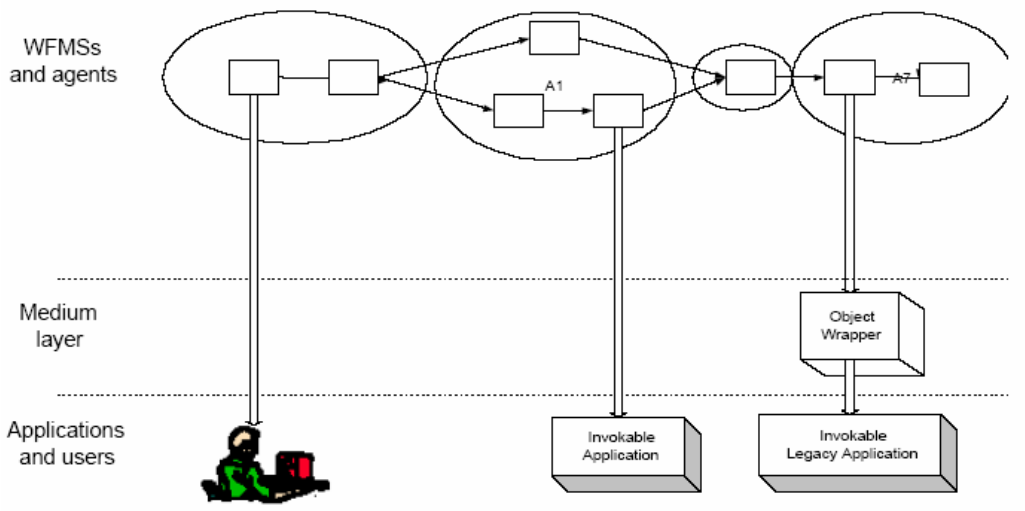

Figure 2.4 Agent-based Workflow Management System

This scenario is quite interesting in real world; the business process is across several units in a company or even spreads across several companies. Thus a central workflow engine is unable to get all the information of the whole business process in order to control it. Through interactions among the multiple workflow engines the whole business process is fulfilled (Yan, Maamar & Shen, 2001).

Agents in this scenario take full responsibilities of a workflow management system, which means agent has all the means to analyze, automate, integrate, and inspect workflows. More important, agent should have means to communicate and interact with each other. Besides that, agents' high-level ability, such as learning, negotiation, can add values to workflow system (Yan, Maamar & Shen, 2001).

Agents in this scenario should be much more complex than the first one. The main features include: autonomous, communication, self-consistent, goal-oriented, and react to environment. Other high-level features such as learning, negotiation are also benefit. But since these techniques are not matured, these features are just promising ones (Yan, Maamar & Shen, 2001).

The usage of agents in this scenario benefits workflow technology in the following ways:

- Providing distributed system architecture. There are several system architectures in multi-agent systems (Shen, 2000), which can be used in a distributed system for implementing workflow management systems.
- Providing communication methods. Agent communication languages are studied quite much. There are several communication languages based on semantic messages, such as KQML. Theses languages can enable interoperation of workflow systems, especially for the heterogeneous ones.
- Providing automation behavior. Agent has the ability to execute tasks on its own without human involvement. Agent also has some decision power according to its goal.
- Reacting to environment. Agent can adjust itself, for example, forming new activity and new routing.
- Benefits of high level features. These high level features include learning, negotiation, and planning. Though these features are not fully implemented for industrial applications, they are what promising.

2.4 Critics About Agent-Based Workflow Management Systems

This study proposes a system which falls in Agent-Based Workflow Management category because of the benefits listed in previous section. Nowadays, none commercial products fall in this category. But several prototype systems exist in research areas. Various system architectures are presented. ADEPT (Advanced Decision Environment for Process Tasks) is an early architecture that focuses on agent based workflow systems (Jennings, 1996). ADEPT focuses on the structure of the agents and multi-agent systems like the frameworks of communication, negotiations and protocols because ADEPT is improved early stages of multi-agent systems. ADEPT does not contain so much design patterns on workflow management but shows a guide line to researchers. Many researchers who are interested in agent based process management systems mentioned the study of ADEPT.

Dagenham (1998), presents an ad-hoc workflow system for processing applications received by a university department from potential research students. In his proposal, agents act as application agent, supervisor agent, control agent, and admin agent. These agents delegate the functions of correspondent users. Another routing agent is to mediate the communication between agents. Debenham's system has several components which do not exist in regular workflows. For example, workers who are taking part in a workflow do not need another actor to communicate each other.

Another kind of system architecture uses mobile agents (Budimac, 1999). Mobile agent represents work item and the mobile agent takes care which of the path the agent should go. On each node a server agent resides, which accepts mobile agents, interface to user, invokes the function condition for each work item agent, and prevent the work item idle in one node. This approach may be good solution if there are a lot of tasks to be done at any time when the resources (humans or agents) are not enough to carry the workload. If system does not contain so much workload, this approach is much more complex than regular workflow systems. Design issues of workflow will have more difficulties and this is not a favorable thing when dynamic environment is considered. FIPA standard suggests using DF (Directory Facilitator), which returns logical addresses of possible agents which supply desired service. Afterwards the task assignment is accomplished by messaging with communicative intentions with the agents. This approach is much closer to the peer-to-peer model.

The Council for the Central Laboratory of the Research Councils (CCLRC) is one of Europe's largest multidisciplinary research organizations supporting scientists and engineers world-wide. One of CCLRC's departments, Business and Information Technology Department (BITD), is studying at an agent based system called Pellucid (A Platform for Organisationally Mobile Public Employees). Pellucid project is concerned with knowledge management for public employees, specifically for those who are organizationally mobile, moving from one department or post to another. Pellucid is developing a customizable software platform for developing knowledge management systems to aid such employees. It integrates several advanced

information technologies, including autonomous cooperating agents; responsive interaction with the end-users; workflow and process modeling; organizational memory; and metadata for accessing document repositories. The platform is a prototype platform. A usable platform for commercial needs is not available yet. The competences of the agent classes are as follows; personal assistant agents, shadow personal assistant agents, environment tracking agents, role agents, task agents, information search and access agents, monitoring agents. Personal assistant agents are responsible of handling task initiation by employee; responding to requests for information; presentation of information both spontaneously and on request. Shadow personal assistant agents are responsible of memory of task performance by former employees.

Environment tracking agents are responsible of tracking processes requiring interaction with the external environment; feeding updates on environment into workflow processes. Role agents are responsible of initiating and supervising tasks within employees' broad roles; anticipation of employees' needs within their roles. Task agents are responsible of generating, supervising and executing processes to achieve particular tasks. Information search and access agents are responsible of locating and retrieving information on request from diverse repositories. Monitoring agents are responsible of monitoring communication and behaviour between individual agents; detection of patterns and trends and storage in organizational memory; dynamic modification of processes.

Concepts which are realized by personal assistant agent and role agents are mandatory concepts for any agent based system. On the other hand, the aims of the other agents except monitoring agent are confusing. Their functionalities can be gathered in role agents. These are done by the employees in daily life of a firm. There are not special employees who perform these functions. Thus, there is no need for the special agents to follow these tasks. Supervisor who deals with system performance can be represented by monitoring agent. This is a useful function. However, it will be a hard work to build an agent like this. This can be achieved easily via storing messages among agents in a central repository. Content language of

messages would be better if it has ability to represent knowledge; like RDF or OWL. The framework presented in this study contains built-in content language in RDF. RDF is especially chosen if any demand of knowledge representation of enterprise arises after a while. Meanwhile the user is not pushed to use RDF. Any language can be used with the framework.

Agent-enhanced workflow systems are widely adopted by commercial products, while the agent-based workflow systems are still in prototypes in research laboratories. However, the agent-based workflow systems are attracting more and more attention for studying the interaction of multiple workflow systems, which are the requirement of next generation of workflow systems.

CAHPTER THREE

REFERENCE TECHNOLOGIES

3.1 FIPA

The Foundation for Intelligent Physical Agents (FIPA) is a body for developing and setting computer software standards for heterogeneous and interacting agents and agent-based systems.

Mission of FIPA is "The promotion of technologies and interoperability specifications that facilitate the end-to-end inter-working of intelligent agent systems in modern commercial and industrial settings."

Based on the first set of specifications released in 1997, at the end of 2002 FIPA finally released the standard. The standard targets interoperability and, as a consequence, it focuses on the external behaviour of the system components, leaving open the implementation details and the internal architectures. FIPA, the standards organization for agents and multi-agent systems was officially accepted by the IEEE as its eleventh standards committee on 8 June 2005.

FIPA standard fully embraces the agent paradigm and, in particular, it defines the reference model of an agent platform and a set of services that should be provided. The collection of these services, and their standard interfaces, represents the normative rules that allow a society of agents to exist, operate, and be managed. Being agents social and needing to communicate, the Agent Communication Language (ACL) is one of the main assets of the FIPA standard.

FIPA standard encapsulates standards dealing with agent platforms, interactions between agents and agent platforms. The agent, itself, is outside of the scope of FIPA standard.

FIPA specification establishes the logical reference model for the creation, registration, location, communication, migration and retirement of agents. The FIPA standard do not attempt to prescribe the internal architecture of agents nor how they should be implemented, but it specifies the interface necessary to support interoperability between agent systems. Because of this, FIPA specified the FIPA Abstract Architecture not a concrete model.

3.1.1 FIPA Abstract Architecture

The primary focus of this FIPA Abstract Architecture is to create semantically meaningful message exchange between agents which may be using different messaging transports, different Agent Communication Languages, or different content languages. This requires numerous points of potential interoperability. The scope of this architecture includes:

A model of services and discovery of services available to agents and other services, message transport interoperability, supporting various forms of ACL representations, supporting various forms of content language, and, supporting multiple directory services representations (Foundation For Intelligent Physical Agents FIPA, 2002).

3.1.2 Agent Management Reference Model

Agent management provides the normative framework within which FIPA agents exist and operate. It establishes the logical reference model for the creation, registration, location, communication, migration and retirement of agents.

The entities contained in the reference model (see figure 3.1) are logical capability sets (that is, services) and do not imply any physical configuration (FIPA, 2004a).

An agent is a computational process that implements the autonomous, communicating functionality of an application. Agents communicate using an Agent Communication Language. An Agent is the fundamental actor on an AP which combines one or more service capabilities, as published in a service description, into

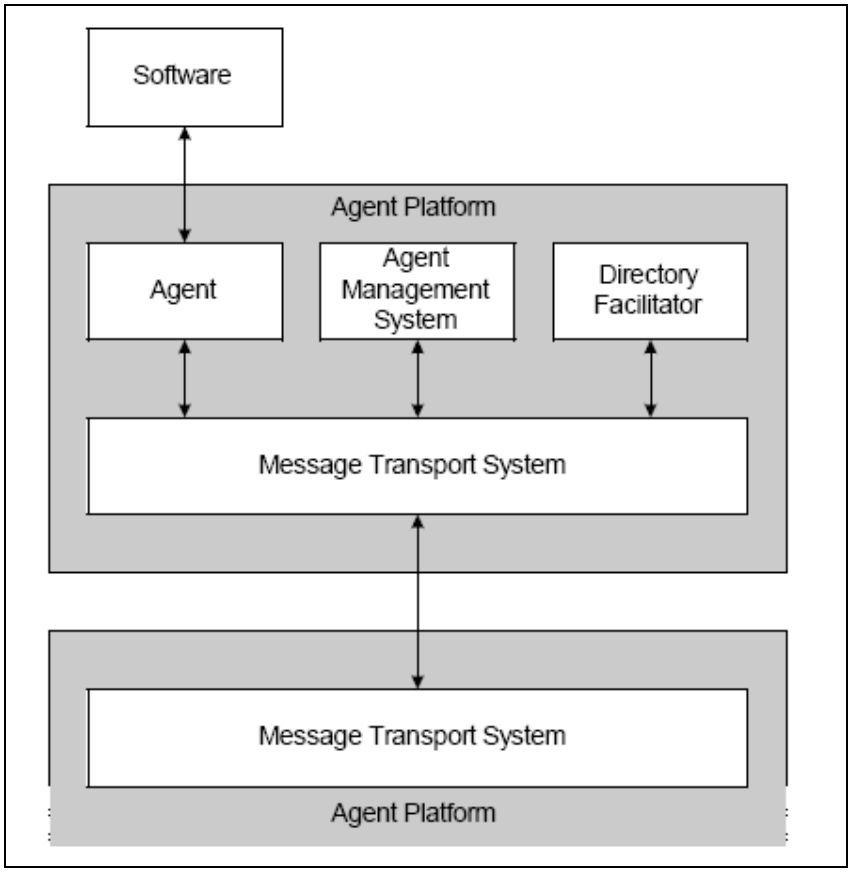

Figure 3.1 Agent Management Reference Model

a unified and integrated execution model. An agent must have at least one owner, for example, based on organizational affiliation or human user ownership, and an agent must support at least one notion of identity. This notion of identity is the Agent Identifier (AID) that labels an agent so that it may be distinguished unambiguously within the Agent Universe. An agent may be registered at a number of transport addresses at which it can be contacted (FIPA, 2004a).

A Directory Facilitator (DF) is an optional component of the AP, but if it is present, it must be implemented as a DF service. The DF provides yellow pages services to other agents. Agents may register their services with the DF or query the DF to find out what services are offered by other agents. Multiple DFs may exist within an AP and may be federated (FIPA, 2004a).

An Agent Management System (AMS) is a mandatory component of the AP. The AMS exerts supervisory control over access to and use of the AP. Only one AMS will exist in a single AP. The AMS maintains a directory of AIDs which contain transport addresses (amongst other things) for agents registered with the AP. The AMS offers white pages services to other agents. Each agent must register with an AMS in order to get a valid AID (FIPA, 2004a).

A Message Transport Service (MTS) is the default communication method between agents on different APs.

An Agent Platform (AP) provides the physical infrastructure in which agents can be deployed. The AP consists of the machine(s), operating system, agent support software, FIPA agent management components (DF, AMS and MTS) and agents (FIPA, 2004a).

3.1.3 Message Structure

The structure of a message is a key-value-tuple and is written in an agent-communication-language, such as FIPA ACL. The content of the message is expressed in a content-language. Content expressions can be grounded by ontologies referenced within the ontology key-value-tuple. The messages also contain the sender and receiver names, expressed as agent-names. Agent-names are unique name identifiers for an agent. Every message has one sender and zero or more receivers. The case of zero receivers enables broadcasting of messages such as in ad-hoc wireless networks (FIPA, 2004a).

The FIPA ACL is based on the speech act theory and on the assumptions and requirements of the agents paradigm described above. FIPA standardized an extensible library of 22 communicative acts that allow representation of different

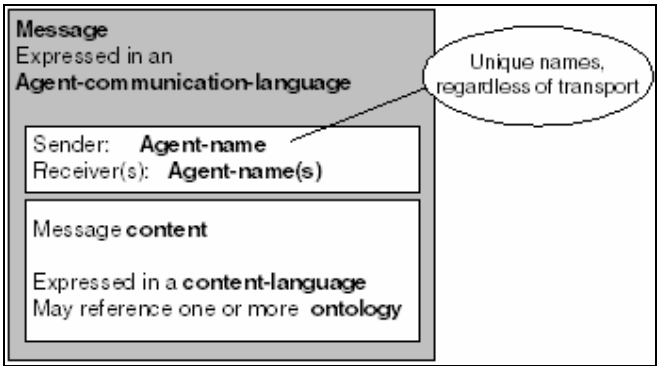

Figure 3.2 A Message

communicative intentions (FIPA, 2004a). These are; “Accept Proposal”, “Agree”, “Cancel”, “Call for Proposal”, “Confirm”, “Disconfirm”, “Failure”, “Inform”, “Inform If”, “Inform Ref”, “Not Understood”, “Propagate”, “Propose”, “Proxy”, “Query If”, “Query Ref”, “Refuse”, “Reject Proposal”, “Request”, “Request When”, “Request Whenever”, “Subscribe”. Ones which are used most are “Agree”, “Refuse”, ”Inform”, and, “Request”.

FIPA also defined the structure of a message that allows representing and conveying information useful to identify sender and receivers, the content of the message and its properties (e.g. the encodings and the representation language), and, in particular, information useful to identify and follow threads of conversation between agents and to represent timeouts for the communication (FIPA, 2004a). Common patterns of conversations have been also defined by FIPA, the so-called interaction protocols, “FIPA-Request”, “FIPA-Query”, ”FIPA-RequestWhen”, ”FIPA-ContractNet”, “FIPA-IteratedContractNet”, “FIPA-AuctionEnglish”, “FIPA-AuctionDutch”, “FIPA-Brokering”, ”FIPA-Recruiting”, “FIPA-Subscribe”, and, “FIPA-Propose”.

3.1.4 FIPA-Request-Protocol

The FIPA Request Interaction Protocol (IP) allows one agent to request another to perform some action. The Participant processes the request and makes a decision whether to accept or refuse the request. If a refuse decision is made, then “refused”

becomes true and the Participant communicates a refuse. Otherwise, “agreed” becomes true (FIPA, 2004b).

If conditions indicate that an explicit agreement is required (that is, “notification necessary” is true), then the Participant communicates an agree. The agree may be optional depending on circumstances, for example, if the requested action is very quick and can happen before a time specified in the reply-by parameter (FIPA, 2004b). Once the request has been agreed upon, then the Participant must communicate either:

- A failure if it fails in its attempt to fill the request,
- An inform-done if it successfully completes the request and only wishes to indicate that it is done, or,
- An inform-result if it wishes to indicate both that it is done and notify the initiator of the results.

Any interaction using this interaction protocol is identified by a globally unique, non-null conversation-id parameter, assigned by the Initiator. The agents involved in the interaction must tag all of its ACL messages with this conversation identifier. This enables each agent to manage its communication strategies and activities, for example, it allows an agent to identify individual conversations and to reason across historical records of conversations (FIPA, 2004b).

3.1.5 Message Transport

When a message is sent it is encoded into a payload, and included in a transport-message. The payload is encoded using the encoding-representation appropriate for the transport. For example, if the message is going to be sent over a low bandwidth transport (such a wireless connection) a bit efficient representation may be used instead of a string representation to allow more efficient transmission.

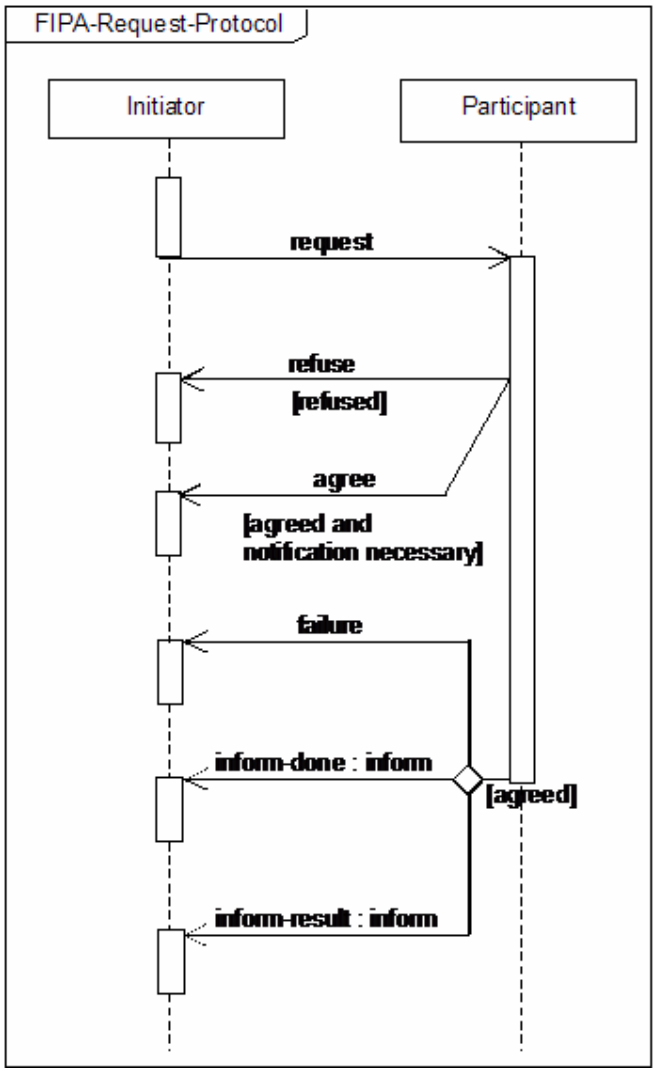

Figure 3.3 Fipa Request Protocol

The transport-message itself is the payload plus the envelope. The envelope includes the sender and receiver transport-descriptions. The transport-descriptions contain the information about how to send the message (via what transport, to what address, with details about how to utilize the transport). The envelope can also contain additional information, such as the encoding-representation, data related security, and other realization specific data that needs be visible to the transport or recipient(s).

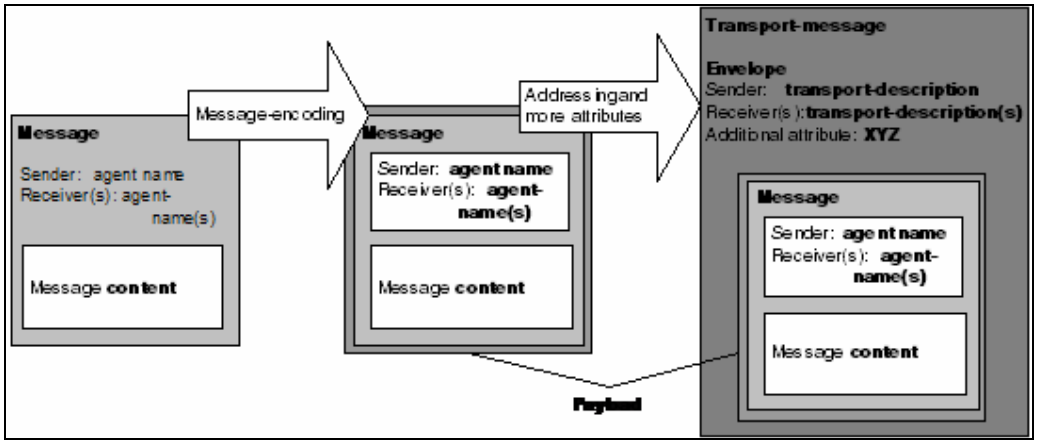

Figure 3.4 A Message Becomes a Transport-message

In figure 3.4, a message is encoded into a payload suitable for transport over the selected message-transport. It should be noted that payload adds nothing to the message, but only encodes it into another representation. An appropriate envelope is created that has sender and receiver information that uses the transport-description data appropriate to the transport selected. There may be additional envelope data also included. The combination of the payload and envelope is termed as a transport-message.

3.2 Resource Description Framework (RDF)

Resource Description Framework (RDF) is a family of specifications for a metadata model that is often implemented as an application of XML. The RDF family of specifications is maintained by the World Wide Web Consortium (W3C).

The RDF metadata model is based upon the idea of making statements about resources in the form of a subject-predicate-object expression, called a triple in RDF terminology. The subject is the resource, the "thing" being described. The predicate is a trait or aspect about that resource, and often expresses a relationship between the subject and the object. The object is the object of the relationship or value of that trait.

This mechanism for describing resources is a major component in what is proposed by the W3C's Semantic Web activity: an evolutionary stage of the World Wide Web in which automated software can store, exchange, and utilize metadata about the vast resources of the Web, in turn enabling users to deal with those resources with greater efficiency and certainty. RDF's simple data model and ability to model disparate, abstract concepts has also led to its increasing use in knowledge management applications unrelated to Semantic Web activity.

A collection of RDF statements intrinsically represents a labeled, directed pseudo-graph. As such, an RDF-based data model is more naturally suited to certain kinds of knowledge representation than the relational model and other ontological models traditionally used in computing today.

Until now, the Web has been designed for direct human processing, but the next-generation Web called “Semantic Web”, aims at machine-processable information.

The Semantic Web will only be possible once further levels of interoperability have been established. Standards must be defined not only for the syntactic form of documents, but also for their semantic content.

XML and RDF are the current standards for establishing semantic interoperability on the Web, but XML addresses only document structure. RDF better facilities interoperation because it provides a data model that can be extended to address sophisticated ontology representation techniques.

3.2.1 Modeling

All items that RDF expressions describe are called resources, and broadly speaking, anything a Universal Resource Identifier can name is also a resource. Consequently, RDF can describe not just things on the Web(such as pages, parts of pages, or collections of pages) but also things not on the Web- as long as they can be named using some URI scheme.

The RDF description model uses object-attribute-value triples: we can view instances of the model as directed or labeled graphs (which resemble semantic networks), or we can take a more object-centric view and think of RDF as a frame based representation system. In RDF, these triples are known as statements.

There are a number of ways to show statements or sets of triples for discussion:

- English-like statements: The simplest way of modeling is using English-like statements such as:

  John Smith is the creator of http://www.example.org/index.html

  More generally we could say that:

  subject has a predicate of object

- Directed labeled graphs: Another way of expressing our statements is to use directed labeled graphs. These are often used in the relational database world- where they are called nodes and arcs and would look like in figure 3.5:

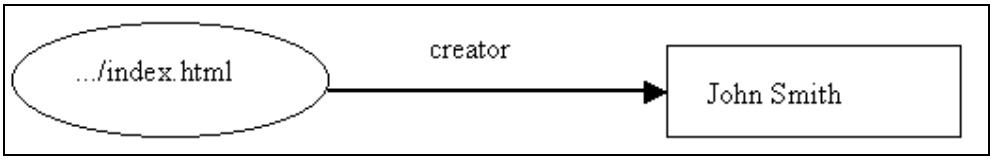

Figure 3.5 Nodes and arcs diagram

- List the three parts of the triple: In this diagram the right-hand side is rectangle rather than an oval – as we saw in the earlier diagram. This indicates that the value is a string literal if the value was a resource then the right-hand side would be an oval. In this layout we simply list the three parts of the triple.

  <http://www.example.org/index.html>

  <http://purl.org/dc/elements/1.1/creator>

  "John Smith".

3.2.2 Making Statements about Resources

The statement “John Smith is the creator of http://www.example.org/index.html” can be modeled in RDF (if John Smith has a URI of http://www.example.org/staffid/85740):

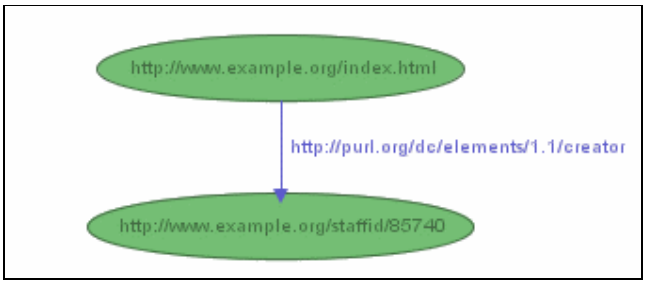

Figure 3.6 Making Statements about Resources

Since the URIref for the creator of the page is a full-fledged resource, additional information about him can be recorded.

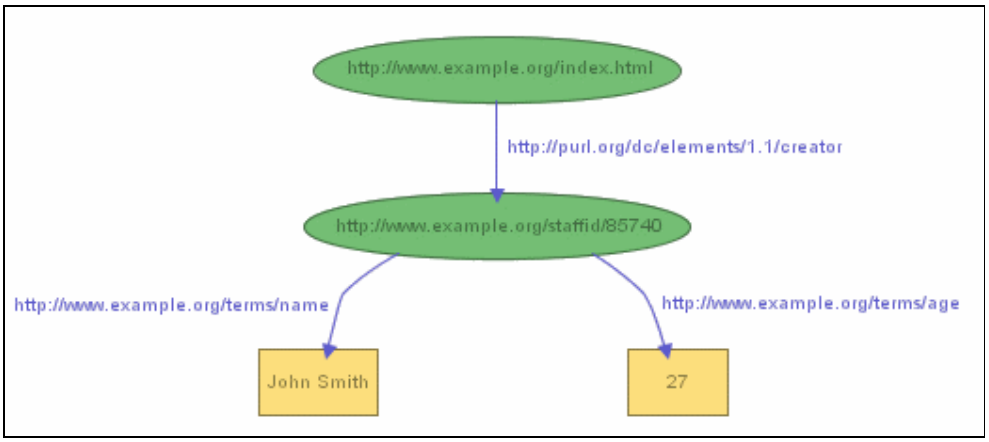

Figure 3.7 Making statements about resources

Using URIrefs as subjects, predicates, and objects in RDF statements allows us to begin to develop and use a shared vocabulary on the Web. But people can still use different URIrefs to refer to the same thing. However, the fact that these different URIrefs are used in the commonly-accessible "Web space" creates the opportunity

both to identify equivalences among these different references, and to migrate toward the use of common references.

3.2.3 RDF/XML Syntax

RDF uses XML syntax for the syntactic representation of model instances. RDF is essentially a data model and does not strive to replace XML. Instead, it builds a layer on top of it, making interoperable exchange of semantic information possible. RDF lacks primitive data types, so strings are essentially the only literals available; XML atomic data types will be used once W3C completes work on the XML Schema.

We often hear claims such as, “You don’t need RDF; you can do everything with XML”. Sure, you could do that, but essentially you would end up reinventing the wheel by building a similar layer on top of XML, that RDF already introduced.

RDF content should appear within an rdf:RDF element. Only one rdf:RDF element should appear within a given page.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
...
</rdf:RDF>
```

rdf:Description is the basic syntax for defining RDF statements.

```
<rdf:Description rdf:about="subjectURI">
<predicate1 rdf:resource="objectURI"/>
<predicate2>objectLiteral</predicate2>
...
</rdf:Description>
```

produces the RDF graph below:

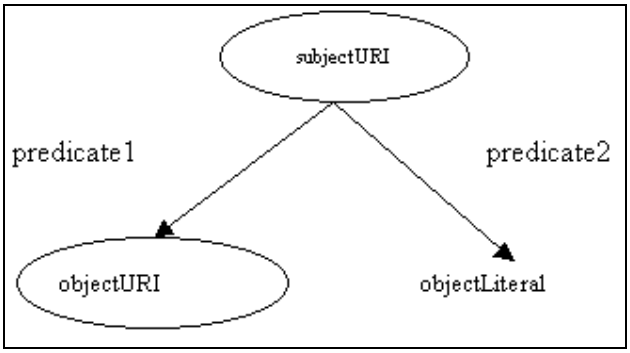

Figure 3.8 rdf:Description

3.2.4 Ontology

A common vocabulary of agreed upon definitions and relationships between those definitions, to describe a particular subject domain.

Agents in multi-agent systems, request services and get responses via messages. In other words, agents are prompted by messages. All agents in the multi-agent systems must know how to prompt each other. When building a multi-agent system, it may be so useful to compose a vocabulary to be used in messages.

The vocabularies may be used in several message types. Composing relationships between messages is easier. When the messages are stored in a big repository, the knowledge representation of the whole system may be accomplished easily.

3.3 JADE (Java Agent DEvelopment Framework)

JADE (Java Agent DEvelopment Framework) is a software Framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that supports the debugging and deployment phases. The agent platform can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine

to another one, as and when required. JADE is completely implemented in Java language and the minimal system requirement is the version 1.4 of JAVA (the run time environment or the JDK) (Jade Group, 2006).

The synergy between the JADE platform and the LEAP (Lightweight Extensible Authentication Protocol (LEAP) is a proprietary wireless LAN authentication method developed by Cisco Systems.) libraries allows obtaining a FIPA-compliant agent platform with reduced footprint and compatibility with mobile Java environments down to J2ME-CLDC MIDP 1.0. The LEAP libraries have been developed with the collaboration of the LEAP project and can be downloaded as an add-on of JADE from this same Web site (Jade Group, 2006).

JADE is free software and is distributed by Telecom Italia, the copyright holder, in open source software under the terms of the LGPL (Lesser General Public License Version 2). Since May 2003, a JADE Board has been created that supervisions the management of the JADE Project. Currently the JADE Board lists 5 members: Telecom Italia , Motorola, Whitestein Technologies AG., Profactor GmbH, and France Telecom R&D.

Jade (Java Agent Development Environment) is a robust and efficient environment for distributed "agent" systems.

It is a good choice because:

- it has all the agent features that is needed (and more)
- it is efficient and tolerant of faulty programming
- it follows FIPA standards
- the user group is very active and implementers typically respond to problems within 24 hours

On the negative side, JADE may be disappointing to AI people because it lacks mechanisms for "intelligence", planning or reasoning. However, the JAVA base

means that JADE can interact relatively easily with Java implementations of Prolog or Expert systems (JESS).

3.3.1 JADE Overview

In a typical Jade application, the various agents in the system are located in many different "containers" executing on several computers; but there is ONE central agent (the AMS) which keep track of all addresses. Similarly, there must be only one DF. The Jade program which runs the AMS and the DF is called the "Main-Container" and it must be started first. To ease interconnection between various containers, Jade makes good use of the standard Java RMI registry facilities (on port 1099). The Main-Container registers with the local RMI registry - even starting one if none already exists - and all other containers will look for its address in that registry. If a secondary container executes on the same computer as the Main-Container, it will connect to the existing AMS and the DF automatically. Only if it runs on a different computer, do we have to specify the host address of the Main-Container. Thus with secondary Jade environments (containers), we will use the following options (Vaucher, & Ncho, 2003):

-container: to indicate that the container is 'secondary' and should use the services of the Main-Container.

-host : to indicate where the Main-Container may be found

Here is an example where we start 4 Jade containers on the same machine: 1) a main container with no other agents, 2) a container with a Simple1 agent called Fred, 3) another secondary container with another Simple agent and 4) finally a container for the RMA agent. Ideally we should start these containers in different windows so that the individual outputs are not mixed up (Vaucher, & Ncho, 2003).

```
java jade.Boot
java jade.Boot -container Fred:Simple1
java jade.Boot -container Harry:Simple1
java jade.Boot -container -gui
```

It is in the nature of agents to operate independently and to execute in parallel with other agents. The obvious way to implement this is to assign a java Thread to each agent - and this is what is done in Jade (Vaucher, & Ncho, 2003).

However, there is often the need for further parallelism within each agent because an agent may be involved in negotiations with other agents and each negotiation should proceed at its own pace. We could use additional Threads to handle each concurrent agent activity but this becomes very inefficient because Java Threads (in spite of the light-weight connotation of the name) were not designed for large-scale parallelism. Rather, they were designed to allow Java programs to exploit the real parallelism of multi-processor architectures and, in current Java releases, each Java Threads requires one OS Thread. This means that passing control from one Thread to another, is about 100 times slower than simply calling a method (Vaucher, & Ncho, 2003).

In order to support efficiently parallel activities within an agent, Jade has introduced a concept called Behaviour (Vaucher, & Ncho, 2003).

3.3.2 Behaviours

A behaviour is basically an Event Handler, a method which describes how an agent reacts to an event. Formally, an event is a relevant change of state; in practical terms, this means: reception of a message or a Timer interrupt. In Jade, Behaviours are classes and the Event Handler code is placed in a method called action (Vaucher, & Ncho, 2003).

Although the use of Behaviours promotes efficiency, it doesn't simplify programming. Consider coding the steps in a negotiation: sending offers, waiting for counter-offers and finally reaching agreement. This activity consists of an alternation of active phases - when the agent decides what to do and sends messages - and passive phases - when the agent waits for an answer. Threads can pause in the middle of execution to wait for messages and continue without losing context. So if we use Threads, the sequence of activities maps directly into sequences of instructions. Not so when we use Behaviours (Vaucher, & Ncho, 2003).

Behaviour actions are methods, executed one after the other by the agent's Thread after events. Like listeners in graphic interfaces, they cannot pause without blocking all other activity [within the agent]. So this is the important thing to remember about Behaviours is that: Each Behaviour execution corresponds to ONE SINGLE instantaneous active phase (Vaucher, & Ncho, 2003).

To implement long-term activities like a negotiation, we have to provide as many different Behaviours as there are active phases in the activity. We must also arrange for them to be created and triggered in the right sequence (Vaucher, & Ncho, 2003).

Jade provides many useful Behaviours which can be extended to model the complex activity typical of real agents. Basically there are 2 kinds of behaviour classes: primitive, like the Simple or Cyclic behaviours and composite ones which can combine both simple and composite behaviours to execute in sequence or in parallel (Vaucher, & Ncho, 2003).

Primitive behaviours are

- SimpleBehaviour: an under-rated basic class that you can extend in various ways and which often turns out to be the best solution when other promising Behaviours are found to have some hidden quirks
- CyclicBehaviour: This behaviour stays active as long as its agent is alive and will be called repeatedly after every event. Quite useful to handle message reception.
    - TickerBehaviour: a cyclic behaviour which periodically executes some user-defined piece of code
- OneShotBehaviour: This executes ONCE and dies.... Not really that useful since the one shot may be triggered at the wrong time.
    - WakerBehaviour: which executes some user code once at a specificied time
    - ReceiverBehaviour: which triggers when a given type of message is received (or a timeout expires).

Composite behaviours are

- ParallelBehaviour: controls a set of children behaviours that execute in parallel. The important thing is the termination condition: we can specify that the group terminates when ALL children are done, N children are done or ANY child is done.
- SequentialBehaviour: this behaviour executes its children behaviours one after the other and terminates when the last child has ended.

Jade provides other Behaviour but we consider them too complex for beginners. They include:

- SimpleAchieveREInitiator
- SimpleAchieveREResponder
- FSMBehaviour

Figure 3.9 depicts how an Agent class can be constructed via JADE framework.

```
public class MyAgent1 extends Agent
{
   protected void setup()
   {
      addBehaviour( new MyFirstBehaviour () );
   }
}
```

Figure 3.9 Creating an agent

Figure 3.10 depicts how a “Behaviour” class can be constructed via JADE framework.

```
class MyFirstBehaviour extends SimpleBehaviour
{
   public void action()
   {
      System.out.println( "--- Message 1 --- " );
      finished = true;
   }

   private boolean finished = false;
   public  boolean done() {  return finished;  }
}
```

Figure 3.10 Creating a behaviour

3.3.3 Structure of A JADE Message

Here is a list of all attributes of a Jade ACL message. As described in the API documentation, Jade provides get and set methods to access all the attributes. We put in bold the ones we use most (Vaucher, & Ncho, 2003).

- Performative - FIPA message type (INFORM, QUERY, PROPOSE, ...)
    - Addressing
    - Receiver
- Sender (initialized automatically)
- Content - This is the main content of the message
- ConversationID - Used to link messages in same conversation
- Language - Specifies which language is used in the content
- Ontology - Specifies which ontology is used in the content
- Protocol - Specifies the protocol
- ReplyWith - Another field to help distinguish answers
- InReplyTo - Sender uses to help distinguish answers
- ReplyBy - Used to set a time limit on an answer

When you create a message, you have to indicate its type - its performative in ACL lingo - and set the content. This is shown in figure 3.11:

```
ACLMessage msg = new ACLMessage( ACLMessage.INFORM );
msg.setContent("I sell seashells at $10/kg" );
```

Figure 3.11 Initialization of ACL Message.

Our message uses the most common performative: INFORM whereby one agent gives another some useful information. Other types are: QUERY to ask a question, REQUEST to ask the other to do something and PROPOSE to start bargaining. Performatives for answers include AGREE or REFUSE (Vaucher, & Ncho, 2003).

Sending message if we know Agent ID (AID) when the agent behavour is like in figure 3.12.

```
ACLMessage msg = new ACLMessage( ACLMessage.INFORM );
msg.setContent("I sell seashells at $10/kg" );
AID dest = new AID( "store1", AID.ISLOCALNAME);
msg.addReceiver( dest );
send(msg);
```

Figure 3.12 Sending message when AID is known

All messages have an attribute which contains the ID of the sender. Thus, we can answer a message as in figure 3.13:

```
ACLMessage msg = receive();
ACLMessage reply = new ACLMessage( ACLMessage.INFORM );
reply.setContent( "Pong" );
reply.addReceiver( msg.getSender() );
send(reply);
```

Figure 3.13 Replying a received message

If we don't know Agent ID, searching the DF (yellow page service) is usual way of discovering new agents IDs which provide needed services.

JADE implements a Directory Facilitator (DF) agent as specified by FIPA. The DF is often compared to the "Yellow Pages" phone book. Agents wishing to advertise their services register with the DF. Visiting agents can then ask (search) the DF looking for agents which provide the services they desire (Vaucher, & Ncho, 2003).

The DF is a centralized registry of entries which associate service descriptions to agent IDs. The same basic data structure, the DFAgentDescription (DFD), is used both for adding an entry or searching for services (Vaucher, & Ncho, 2003).

For agents to interact usefully in open systems, it is imperative that they use the same language conventions and the same vocabulary. The DF entries thus concentrate on listing the ontologies, protocols and languages which are supported by the agents. Additionally, entries have sets of services which are characterized by a name and name-value properties as well as the ontology/language/protocol conventions they support. Figure 3.14 is the structure of a DF Agent description. It appears quite complex, but most fields are optional and in actual practice we only need to use 1 or 2 attributes (Vaucher, & Ncho, 2003).

Most of the attributes are sets of because an agent could handle messages in a variety of formats; this is reflected in the naming conventions for methods which most often start with add instead of set, for example: addOntologies(String onto).

3.3.4 Interaction Protocols

FIPA specifies a set of standard interaction protocols, which can be used as standard templates to build agent conversations. For every conversation among agents, JADE distinguishes the Initiator role (the agent starting the conversation) and the Responder role (the agent engaging in a conversation after being contacted by some other agent). JADE provides ready-made behaviour classes for both roles in

```
DFAgentDescription
        Name:     AID                  //  Required for registration
        Protocols:   set of Strings
        Ontologies:  set of Strings
        Languages:   set of Strings
        Services:    set of {
         { Name:   String   //  Required for each service specified
           Type:   String     //  Required ...
           Owner:  String
           Protocols:   set of Strings
           Ontologies:  set of Strings
           Languages:   set of Strings
           Properties: set of
            { Name:   String
              Value:  String
            }
         }
```

Figure 3.14 The structure of a DF Agent description

conversations following most FIPA interaction protocols. These classes can be found in jade.proto package. They offer a set of callback methods to handle the states of the protocols with a homogeneous API. All Initiator behaviours terminate and are removed from the queue of the agent tasks, as soon as they reach any final state of the interaction protocol. In order to allow the re-use of the Java objects representing these behaviours without having to recreate new objects, all initiators include a number of reset methods with the appropriate arguments.

Jade has classes called AchieveREInitiator, and, AchieveREResponder for FIPA Request-Protocol. “AchieveREInitiator” class is extended and “handleAgree”, “handleRefuse”, “handleInform”, “handleNotUnderstood” methods are overridden to create a behaviour for the agent in initiator role (see figure 3.3) .

AchieveREResponder class is extended and “prepareResponse”, and, “prepareResultNotification” are overridden to create a behaviour for the agent in participant role.

CHAPTER FOUR

SYSTEM SOLUTIONS FOR BUSINESS PROCESSES

This chapter encapsulates the main problem definition of this study. Besides, analysis of the problem in different ways and a solution model are also discussed in this chapter.

4.1 Problem Definition

Tasks in firms are done by various workers. When these tasks are done, the workers request services from each other. The requests and replies often result in transaction data. These transaction data helps the workers to decide. For decades, these data has been stored in digital environments where achieving and reporting them is easier. Some of advanced digital systems support their users when they are taking decisions. These are called decision support systems. These systems are very helpful for middle and upper managers in firms. However, this capability of these systems is rarely used by other workers who are not managers because the decisions other than managerial decisions are determined by couple of situations. These decisions are never considered as unimportant because they are taken everyday in numerous times. When these decisions are taken, usually some standard procedures are followed. These procedures are usually shown in workflow diagrams. Most of the firms have workflow diagrams because they have or going to have various ISO certificates.

Workflow can be defined as the movement of documents and/or tasks through a work process. More specifically, workflow clarifies the questions like: how tasks are structured, who performs them, what their relative order is, how they are synchronized, how information flows to support the tasks and how tasks are being tracked. Workflows are very useful when the tasks are predefined and the relationships between the tasks are well known.

Workflows consist of nodes like transactions, tasks, and decisions. These nodes are usually atomized. They can not be divided into smaller nodes. Tasks and transaction nodes are so definitive and decisions in workflows are straightforward. Decisions are usually taken by checking some values. These values indicate the environment of the firm and are usually stored in database systems. According to decisions actions are taken. These actions are taken numerous times in a work day. These works become a noteworthy workload for workers. A system that can decide and take actions in some circumstances on behalf of the workers would be so beneficial. This would take a great deal of the workload from the workers. A system can not be thought without human input. However, a computerized system which need a few input can be built easily because workflows usually consist of straightforward steps. Besides, the steps in workflows fit the nature of computerized system because these steps are taken numerous times. The computers are very good at iterative works as everyone knows.

In a summary, the problem definition is how a system that follows workflows on be half of the workers should be designed.

4.2 Analysis of Workflows

The idea of workflow is to bring processes, people and information together. This section will try to analyze these three components of workflows shortly.

Process identifies the work to be done. Process may not be so definitive which steps should be taken through the work. It may consider the main perspective of the work and the goals to be reached at the end of the work. Workflow can also be considered as a process. However, it is so definitive that every work step is defined. Workflow can also consist of some sub-processes.

More than one participant, person (hereafter we will refer to the participants as actors) exist in a workflow. The steps of workflows are under responsibilities of the actors.

Actors request services from each other. Requests and responds among actors are often called as transactions. Transactions hold data. When these data come together, they turn into information. Besides, information from domain databases is added into workflows by queries of actors. Figure 6.1 shows a workflow for ordering a product.

Another key property is the goals in workflow systems. Every actor has individual goals in workflows. Some of them are conflicting goals, some of them not. For example, assume Goal-1 is the goal of sale representative to get best delivery date from production planner. Goal-2 is the goal of production planner to satisfy all sale representatives.Goal-3 is the goal of production planner to check for the production line work in a most efficient way. Goal-1 and Goal-2 are analogous but Goal-2 and Goal-3 are conflicting. All these goals are considered when a workflow is designed.

4.3 Analysis of Legacy Systems

Organizations tend to focus only on data and data management. This comes up with a data layer and a layer of application domains over data layer (see Figure 4.1).

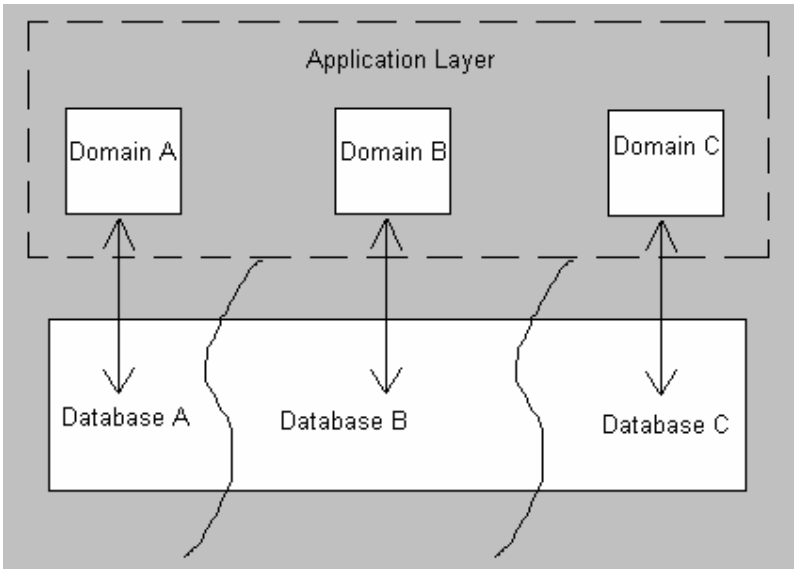

Figure 4.1 A Legacy System

Managing a workflow needs inter-communications among application domains. When a need of intercommunication among applications arises, legacy systems tend

to communicate over data layer. In fact, the business process in real life is going on the basis of request-response manner. Person in one domain requests a service from another person in another domain. He/she never manipulates other's data where most legacy systems solve intercommunication problems in that manner (see figure 4.2). Some others come up with a solution like building intermediate data layers, such as views in database management systems, to communicate (see Figure 4.3). Such solutions' way is not the way in real business life.

Solving integration problems like the two ways above makes the domains too dependent to each other. If one domain changes its database for a minor need, all other domains must be revised if they will be affected of this change. If they will be affected, all the domains should be adjusted. If a system has been built like a system in Figure 4.3, there will be some independency. However, if the base table of the view has been changed, it will affect the other. In every case, there will be lots of neat work to do.

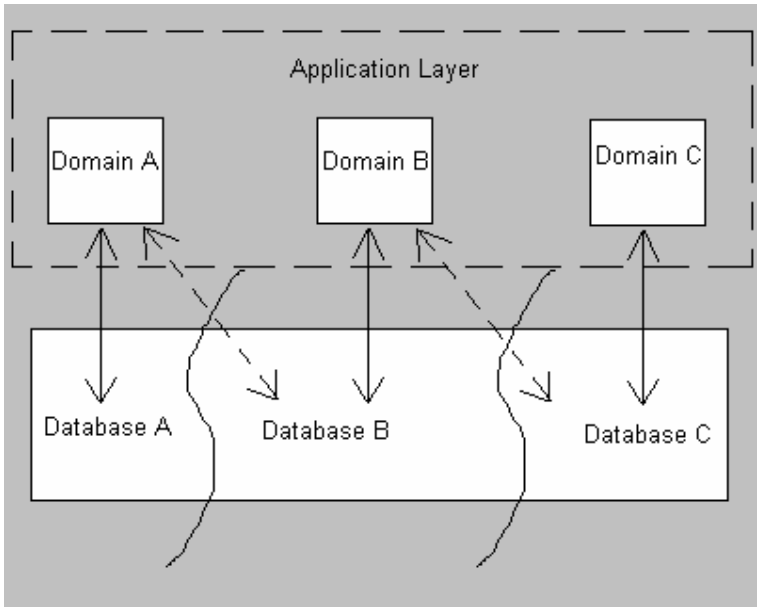

Figure 4.2 Application Domain A reaching database area of Domain B and Application Domain B reaching database area of Domain A.

Companies tend to run their business in a process-oriented approach. Process-oriented approach never makes distinction between work done by a human and a

computer. It concentrates on the actions should be taken for the goal. Software solution should let the designers think in that manner. Having a tool that solves problems in the way people solves in daily life is more realistic approach than the opposite. Legacy systems tend to solve problems in domains. They present solutions for departments. They don't focus on inter department problems and communications.

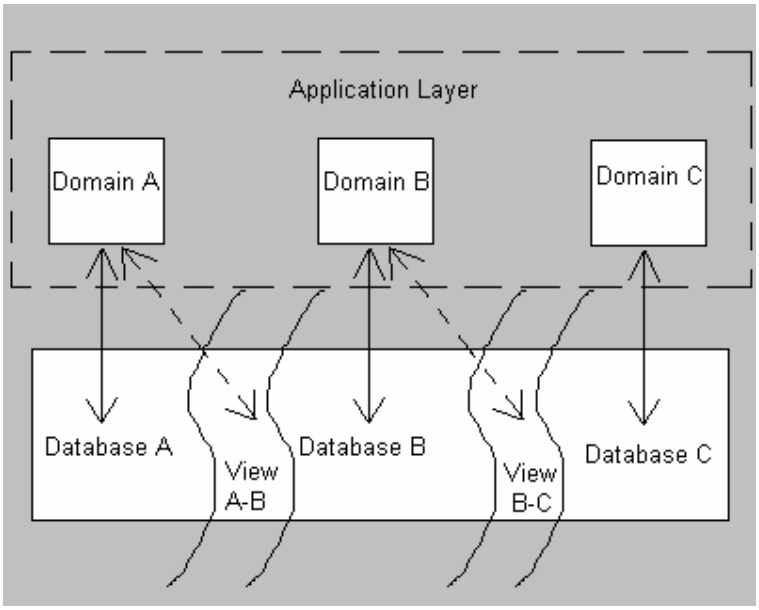

Figure 4.3 Application Domain A communicates with Application Domain B via Database View A-B area and Application Domain B communicates with Application Domain C via Database View B-C area.

4.4 Solution Model: Agent Based System

Many companies have business processes that are unique to its business model. Because these processes tend to evolve over time as the business reacts to market conditions, the software solution must be easily adaptable to the new conditions and requirements. Because workflow is the essential part of the system and workflow force firms to automate process vertically, communication of departments is so important.

Legacy systems have served for companies for long times. Organization can not throw them away easily even if they don't meet new needs. Because so much funds and time are spent to them, a legacy system gets the name of "legacy". But dynamic needs push the firms to change their software system solutions as well as their business procedures. One way of solution is to evolve present systems using legacy system technologies. However, it will lead more funds are spent into legacy systems. This solution will not be long term solution. There will be a need of more funds into legacy systems soon. Besides that, it will be harder to leave them.

Using new technologies without throwing present systems will be the most appropriate choice to firms which don't want to take risks of new systems. This can be accomplished via wrapping legacy systems with new systems with needed technologies.

Agent based systems have a great ability of wrapping systems. Agent based systems focus on messaging among application domains. They don't bring new approach for the application domains. Legacy systems can do what they have done before. Newly constructed agent based system can handle the inter communication of application domains. Figure 4.4 shows the system proposed.

According to analysis, a solution model based on an agent based system is concluded.

If we go back our main issue of how a system that follows workflows should be designed. According to the proposed model, agents are actors in workflows. Every actor in a workflow is represented by an agent. The requests and replies among actors are handled by messages (these messages will be ACL messages in FIPA compliant agent systems, JADE.). Tasks, duties and goals of agents are realized by behaviours (in the sense of JADE framework). Behaviours use domain applications abilities when an action must be taken.

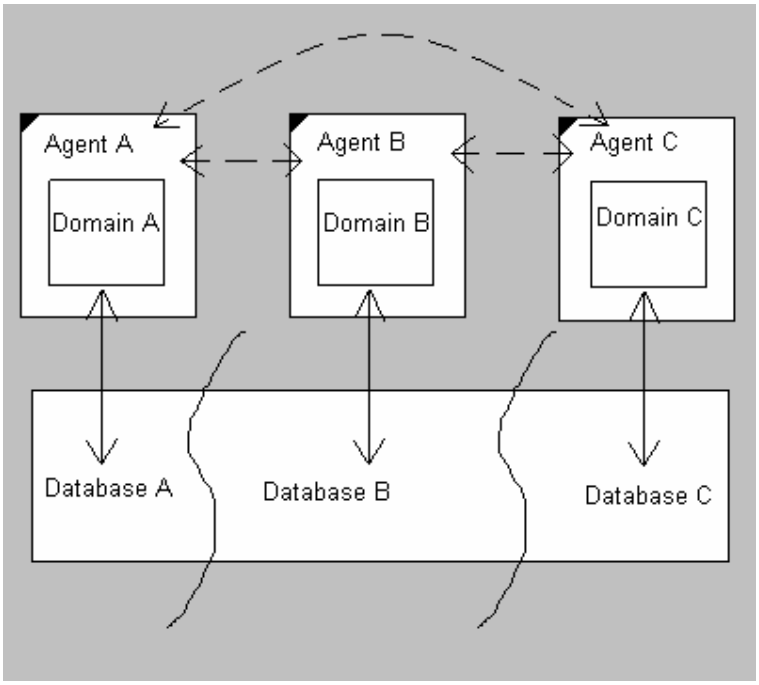

Figure 4.4 Agent based system over legacy system

4.5 A Method For Pre-Designing A Workflow for Agent Based Systems

The followings are proposed steps that should be followed when designing the solution model for a system. If a regular workflow diagram is ready before study, it is easy to follow the steps mentioned below.

- Step-1 Define process
- Step-2 Define actors
- Step-3 Define duties of each agent
- Step-4 Define relationships among actors
- Step-5 Define data structure of messages delivered among agents
- Step-6 Define each agent's goal in the workflow
- Step-7 Define computing steps taken in agents

4.5.1 Step-1 Define The Name of The Process

This step involves definition of the problem. The problem is the name and description of the process as to our model. In this step, the workflow which is going to be handled by agents is chosen. Give a name for the workflow and a brief description.

4.5.2 Step-2 Define Actors

Define actors in the workflow to identify the agents. Each actor will be represented by an agent.

4.5.3 Step-3 Duties of Each Agent

Define which agent will provide which service in this step. This will determine which types of messages will be responded by which agents.

4.5.4 Step-4 Define Relationships Among Actors

Find an answer for the question; which agent can ask for a service? Some agents in the system can't ask for a specific service from an agent. Because services are not only services, they also take actions and they manipulate the environment of the system. For example, only a sale representative agent can place a product order. The designer agent of a product can not place an order of its product.

4.5.5 Step-5 Define Data Structure of Messages Delivered Among Agents.

Define what information must be given to service provider agent to ask for a specific service. Define the data should be in response of the request. For example, order number and set of products must be in the message of order declaration to production planner.

4.5.6 Step-6 Define Each Agent's Goal In The Workflow

Define the goals of actors in workflow. Some of them are conflicting goals, some of them not. If a workflow diagram preexists, it is said that the goals are almost predefined.

4.5.7 Step-7 Define Computing Steps Taken In Agents

Define internal actions to be taken for each agent. Define circumstances in which agent decides and take actions. Define circumstances in which agent must leave the decisions and actions to user. For example, agent can place order under a certain amount of worth. Only users of the agents can place orders over a certain amount.

CHAPTER FIVE

A SIMPLER FRAMEWORK FOR AGENT BASED SYSTEMS

As mentioned in chapter four, a process oriented system needs communication ability among application domains. Chapter four depicts main characteristics of the system that should be built. Besides that, it is concluded that an agent based system will be our solution system. Issues about pre-designing of the agent systems are also presented in that chapter.

Chapter five will propose a framework called SADE (Simple Agent Development Environment) for agent based system. This framework is fully implemented in Java language over JADE Framework. JADE simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications. More details about JADE can be found in chapter three.

Jade embodies extensive experience in the implementation of large Agent systems. However, this maturity leads to difficulty for learners. Many Jade features deal with sophisticated matters that beginners either don't need or don't understand. Similarly, most documentation (reports and Javadoc) is suitable for experienced users but quite inadequate for learners. The examples that come with the distribution are fairly long. System administrators in firms, which are not in IT sector, don’t have so much time to deal with design issues about multi-agent systems. The framework introduced in this chapter aims to ease the design process of multi-agent systems.

Another purpose of the framework introduced in this study is to put the system designers into a methodology discipline. It is also aimed that the designers focus on the problem and avoid them thinking of design issues so much.

5.1 Framework Elements

Basic element of SADE framework is ofcourse agent. Actors in workflows are represented by agents. There are two main agent types in SADE. One of these is called “Core Agent” and the other one is “Gui Agent” (“Graphical User Interface Agent”). “Gui Agent” is responsible from being an interface when the user input is needed into the system. Users of the agents are the actors in the workflows. System asks users to decide when a user decision is needed. This is accomplished by Gui Agents. When the users need to start a process, they use Gui Agents again. Gui Agent only shows the assignments of its user and transmits the input messages from user to responsible Core Agent. In this sense, every Gui Agent has one only one Core Agent. Core Agent carries the procedures of the processes. Workflows lie in Core Agents. Every agent in workflow contains their individual inner steps and communication steps that should be taken. Every agent knows what to do as humans in a process. They don’t know the inner steps taken in other agents. They know only the services can be got from other agents. SADE framework falls into Agent-based Workflow Management System category (see figure 2.4 in literature review chapter).

To get the big picture, it should be started from designing the message structures. So that it can be figured out that every actor roles in the workflow. SADE framework let the designers to design the structures in property-value couples. There is an object called “ObjectModel” in SADE framework. It can hold property-value couples in types, “String”, ”Integer”, ”Long”, ”Duoble”, ”Date”, ”ObjectModel”, and “ObjectBag”. “ObjectBag” object holds set of “ObjectModel”. There is no order among “ObjectModel”s in “ObjectBag”.

“ObjectModel” holds an object type “EncoderDecoder”. This class does the job of converting “ObjectModel”s into representation form of content part of ACL messages and vice versa. As mentioned in related title in Chapter three, an ACL message holds metadata about message and message content. “ObjectModel” object does not ease only the creation of message content from objects but also conversion from ACL message to object. The “ObjectModel” name is given instead of

“MessageModel” because the objects of this class are also used as data structures in methods like retrieving data from databases and others.

Another essential part of SADE is handling the messages income and outcome. Basically, request and respond architecture is used in regular workflows. Because of that, the Request- protocol of FIPA is chosen as main transaction model. Jade framework implements this protocol via AchieveREInitiator, and, AchieveREResponder classes. These are easy to use. However, when a task needs multi requests from other agents, it is getting harder to use these classes. Because of that, these classes are extended and give simple names as Requester and Responder respectively. The objects constructed from these classes request a FlowController object.

FlowController holds a StepGroup object. The StepGroup objects encapsulates the steps which the agent should follow in its part of workflow. It holds inner actions to be taken and communication actions with other agents. StepGroup object holds objects implementing Step interface (in the sense of Java language). Basically, Step interface push the concrete classes, which implement the interface, to define the action in step and next step to be executed. Eventually, every different type of action must be implemented in different concrete Step classes.

FlowController is an interface for following the steps in right order. Jade framework’s action methods of behaviour classes can not be interrupted and resumed after a while. However, agents must wait for responds from other agent to continue the steps that agents should take. There is not an easy way to accomplish ‘wait and resume’ action in Jade framework. FlowController handles ‘wait and resume’ action between requesting a service and receiving a respond.

5.2 Details of SADE Framework

To ease the construction and initialization of agents (core agent and gui agent of domain), a property file must be located inside the folder with agent classes. The property file contains core agent name, gui agent name, JDBC connection

parameters, ObjectModel dictionary class name and package, a property called SomeOptionGettersOfAgentGUI, and another property called TimeOutForGuiMessageResponds which specifies the time will be waited after sending a message from gui agent. JDBC connection parameters , which are not mandatory, are for database connectivity. ObjectModel dictionary is for encoding and decoding ACL messages. SomeOptionGettersOfAgentGUI which holds class name and package is for defining which messages can be sent to which agent by gui agent.

Gui agent of a domain is created by extending AbstractGUIAgent class of SADE framework. Gui agent does not need anything other than construction because behaviours of gui agent are defined and limited. Gui agent, sends the message written from user to its Core agent. Core agent takes care of the message. It holds the knowledge base. If core agent does not respond in time, (response time is specified in agent property file) Gui agent inserts the message into its inbox with a related message which tells the reason that core agent did not respond in time to warn the user. Each GUI agent has a graphical user interface like figure 5.1.

Inbox Table lists the messages has received from agents. Inbox Related Table lists the messages that are related with the selected message in Inbox Table. These related messages are very useful. Core agents sometimes do some part of tasks and gets some intermediate data from other agents but can not get decision. In these circumstances, agent leaves the decision to user. The intermediate data which are composed of multiple messages is put into Inbox Related Table. So the user does not need to collect the data once more. When any message is focused by a left mouse click, the detail of message is shown in Inbox frame. Message can be sent from Outgoing frame. Message Type inbox in Outgoing frame clarifies the type of the message. Message Inbox is followed by an inbox for recipient agent. Send button is for sending the message, naturally. Add Row button is shown up when message contains a list of one type of data should be entered. Refresh button at upper left refreshes the inbox table whether any new message has come. Delete button deletes the message selected in Inbox message with the related messages. Every message in

tables is kept with unique URI address. Messages in inboxes are kept in a RDF file as RDF triples. That's why messages have URI addresses. Other columns of a message in inbox are the agent id of sender, the title of message, and date and time so-called message received.

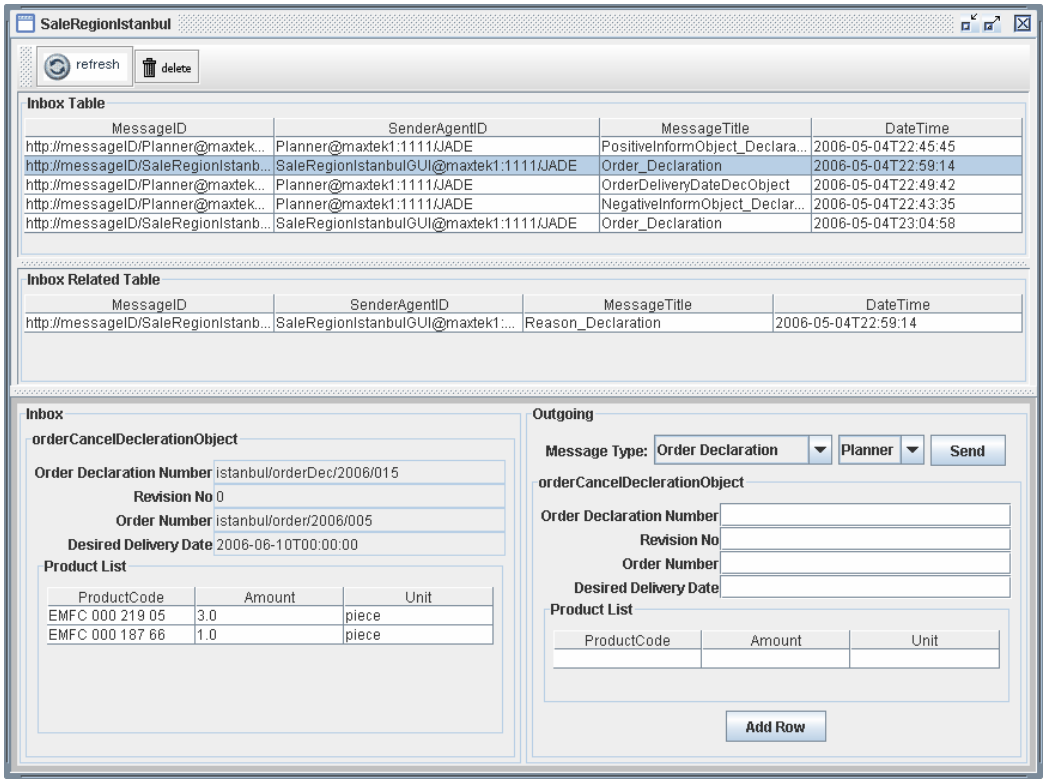

Figure 5.1 Graphical user interface of Gui Agent

Core agent holds the processes as mentioned before. Extended JADE behaviours which holds StepGroup objects takes the actions that should be taken in agents. SADE behaviours are OneShot, Requester, Responder and Ticker SADE behaviour objects. These classes are extended from JADE classes. The main characteristics of the classes come from JADE classes. SADE extended classes add the function of using StepGroup objects.

SADE OneShot behaviour extends JADE OneShotBehaviour. OneShotBehaviour is behaviour that completes immediately and whose action method is executed only once. SADE Ticker behaviour extends JADE TickerBehaviour. It is a behaviour that

never completes and whose action executes the same operations each time it is called. SADE Requester and Responder extend AchieveREInitiator and AchieveREResponder classes respectively. Requester class handles the operations at the initiator part of the FIPA-Request protocol. Responder class handles the operations at the participant part of the FIPA-Request protocol (see figure 3.3).

5.2.1 Execution Steps of SADE Behaviours

SADE behaviour classes hold StepGroup classes which hold the operations that should be done. For example, a StepGroup conceptual informal drawing that is for Requester behaviour may be as follows. In this section the StepGroup classes will be explained conceptually, that is why the node names in figure 5.2 may not be the same ones in the SADE framework. To use the exact names used in SADE framework may cause the reader to misunderstand. The concepts mentioned here are fully realized in SADE framework.

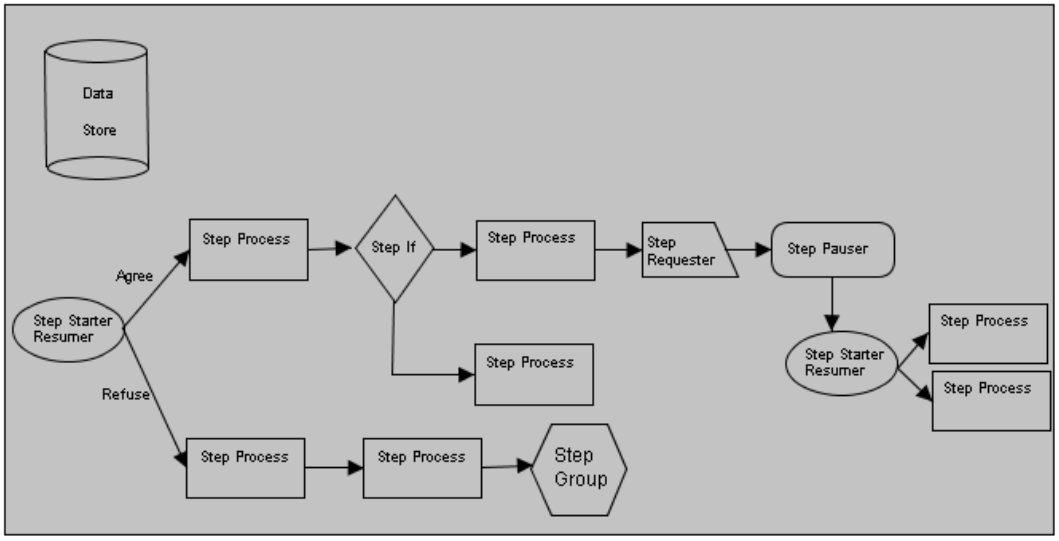

Figure 5.2 A StepGroup informal drawing

A Requester object gets an ACLMessage that will be sent as parameter. When the reply comes, it follows steps according to the income message. Figure 5.2 illustrates a StepGroup which handles a response message of which performative is Agree and Refuse. Every StepGroup object must start with a StepStarterResumer object. This

object identifies the next step according to the message has come. Basically there are five types of steps. They are Step Starter Resumer, Step Process, Step If, Step Requester, Step Pauser. Steps in Step Process type have simply action methods which are executed. They are inner actions that must be taken in agents. Steps in StepIf type have a method that checks the condition that is declared to them. The check method returns a true value or false value, and the value identifies the next step to be executed. Steps in type of Step Requester initiate a new Requester object. New Requester object waits until this Requester pauses. The pause action is made by steps in type of Step Pauser. When an answer comes, Requester object runs the steps where the StepGroup has paused.

The second Step Starter Resumer node is the step where answer message has come. As it is clear, two Requester behaviour objects are taking place in the Step Group at Figure 5.2. Step Groups can also contains other sub Step Groups. The sub step groups are be called and executed in the same manner. Only one type of figure left which has not been mentioned till now in figure 5.2. This is Data Store. As it can be understood from the name of Data Store, it stores data for the Step Group. Steps usually create intermediate data or need variables which hold the state of the execution process. Data Store serves for the steps by holding any type of data which steps can only store. It also holds the messages income and messages to be sent. SADE behaviour classes get the messages to be sent from Data Store. Data Store stores data in two fashions; globally and privately. Global data can also be reached from the sub Step Groups. They can be read and updated by the sub Step Groups. Sub Step Groups can also declare global data and store in Data Store. Private data can only be reached by the steps in same Step Group. A data set by a Step in Step Group A can not be read or updated by a Step in Step Group B. This prevents the confusion of same variable names in other Step Groups accidentally. A global data which holds information for the Sub Step Groups can be set from any main Step Group or Sub Step Group.

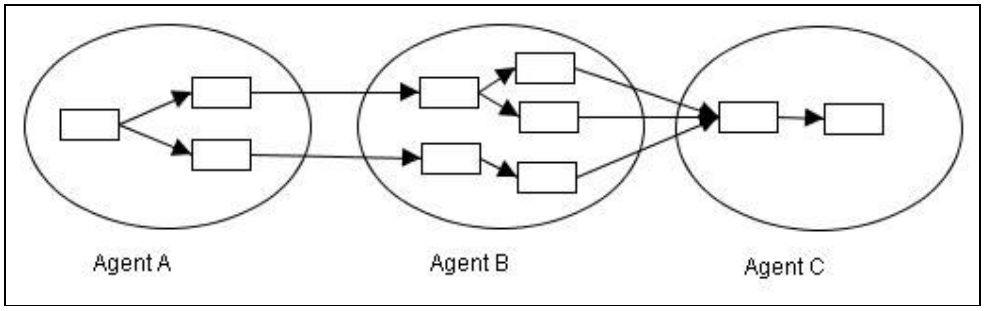

Figure 5.3 Steps taken by in separate agents.

In many firms, the descriptions of jobs are defined by workflows. Step Group concept realizes the steps taking place inside an agent. Workflow diagrams depict the inner operations of all agent and communication operations among actors in one diagram. The agent paradigm does not cover the operations held in separate agents. Step Group concept realizes the inner action that is taken by an agent and communication part of its agent. Figure 5.3 tries to illustrate the steps in agent. The counter part of the communication operation is held in other Step Group in another agent. However, the frame gives chance to designers to think in the same fashion of workflow diagrams. Designer writes the workflow steps as its natural way; step by step. This is realized by ready objects which encapsulate the step types mentioned in this title. All these ready objects implements “Step” interface. “Step” interface defines an execution unit that must be in workflow. Designer needs to initialize the proper object and connect it into other steps in Step Group. This study does not include any graphical user interface for designing workflows. However, because the execution units, “Step” instances, are defined as objects, it is not so hard to add a designing tool into framework. This tool is considered to be implemented by RDF statements but this is out of scope of this study. But SADE framework has a simple tool for ready Step Groups to ease the debug operations while in development of Step Groups. The tool gets a Step Group and outputs a simple RDF model represents the Step Group. This RDF model can be viewed in any RDF viewer tool.

5.2.2 SADE Behaviour and Step Class Diagrams

Figure 5.4 shows the first class diagram of SADE framework. It contains two agent type of SADE, AbstractCoreAgent and AbstractGuiAgent. They are abstract

classes, concrete classes extend these classes and then people get use of them. AgentFactoriesConcrete class gets the property file located in the concrete agent class folder and initiates the agent object. SADE agent object may have one or more JADE Behaviour classes. There is not an obligation like a SADE agent must use a SADE Behaviour. But to get the benefit of the StepGroup class functionalities, SADE Behaviour classes must be used. StepGroup functionalities are got used via

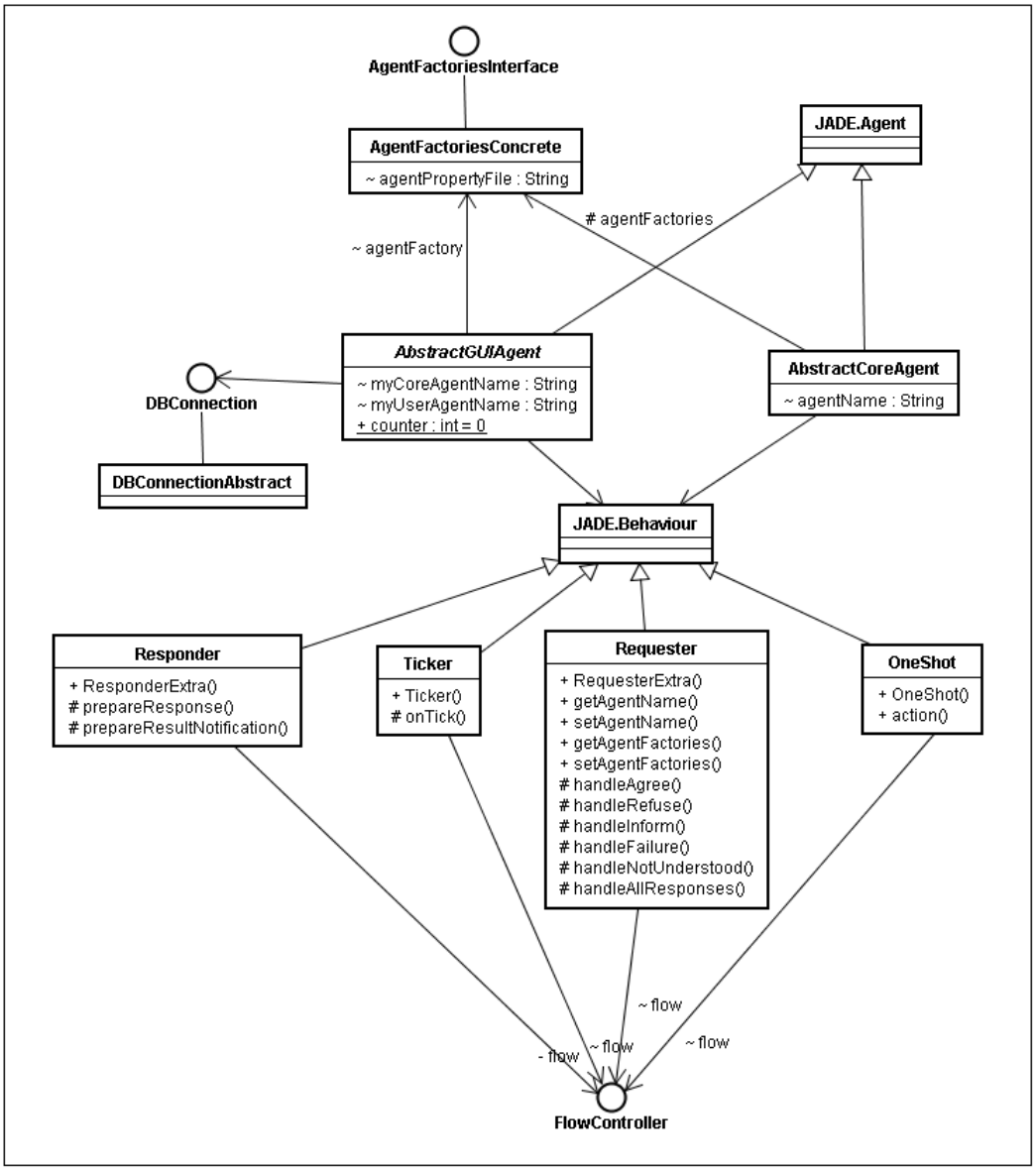

Figure 5.4 SADE class diagram part one

FlowController interface. Interfaces are used when designing SADE framework because interfaces provide great flexibility. When different needs arise after or while designing, interfaces provide the designers to change the attitudes of system without manipulating other parts of the system.

Figure 5.5 shows the second main part of SADE framework, related with Step Group concept. Classes which implement FlowController class contain StepGroup objects and control them. An interface class exists as every time. DefaultFlowController class can be constructed and used via FlowController interface for flexibility. A StepGroup object has a DataStore object and a first step object handle (in the manner of Java language). Most of the abilities of the system described in this chapter are realized in AbstractStep class. The abilities like finding next Step to be executed, setting a data into Data Store, and debugging the Step Group are mainly solved in the scope of this class. But SADE users can not construct an object from AbstractStep directly because it is abstract class. Usable concrete Step classes are at the bottom of the class diagram in figure 5.5. They extend the AbstractStep class so they leave the responsibility of main abilities to AbstractStep and differentiate the actions to be executed. AbstractStep class pushes the implementer classes define their action() method. If users of SADE framework are not satisfied with the actions of ready usable Step classes, they can extend AbstractStep with an action method they need. This architecture gives a great flexibility to the framework. Note that the figure shows some concrete Step classes in SADE framework. It would be a confusing diagram if all of the Step classes are shown.

Other important part of the SADE framework is the part that handles the messages sent among agents and their reflections in agents. Messages sent among agents are not only messages. They contain very useful data for agents. They are data in the form of messages. That's why the name ObjectModel is chosen for them. ObjectModel objects are stored in DataStore object as messages and also data. Every ObjectModel object has an object of EncoderDecoder class. EncoderDecoder object is responsible for converting the data into message and vice versa. EncoderDecoder

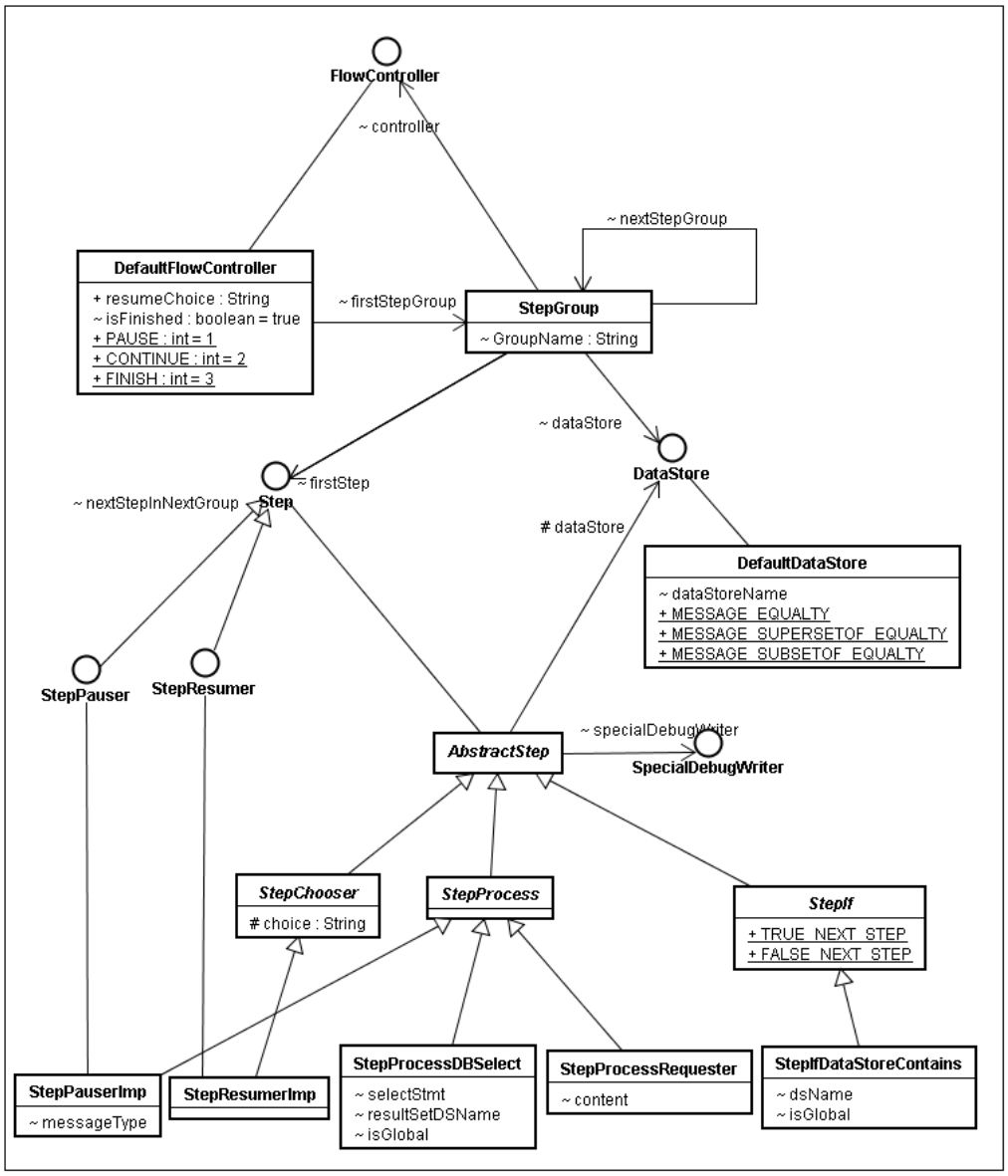

Figure 5.5 SADE class diagram part one

class is not a concrete class. So that, concrete classes like RDFEncoderDecoder class in figure 5.6 extend EncoderDecoder class. So user of SADE framework is not limited about content language of the ACL messages. ObjectModel holds property-value couples in the form of DataAttribute objects. DataAttribute object can hold a Date, String, integer, double, or long value, ObjectModel or ObjectBag object. ObjectModel stores the DataAttributes objects in a hash map with property names. The method getDate() with a proper property name is used to reach the value of a

property in Date format. The other methods are shown in Figure 5.6. ObjectBag objects stores a set of one type of ObjectModels. It can be considered as an array of ObjectModel objects. Different types of ObjectModel objects can not be stored in ObjectBag objects like arrays. ObjectModel class is an abstract class. So that, SADE users must extend the ObjectModel class and define the attributes which will be stored in them. ObjectModel can hold the SQL scripts. Because the data values are kept in them, some special Step objects use the ready scripts in ObjectModel objects. So that, there would no need to retrieve the values of attributes in ObjectModel and form an SQL script. The SQL scripts are formed inside the ObjectModel with the current state of object in execution time.

Figure 5.7 shows the main message structure of SADE in RDF format. The figure illustrates the graph model of RDF statements. Every ObjectModel class has to identify the main RDF node of its. In this example, the main node is "http://www.company.org/order/orderDeliveryDate/deneme". Every ObjectModel class must also identify its context. The node labeled with "j.0:OrderDeliveryDateDecObject " is the context of the message. When any ObjectModel is encoded in RDF, the "j:0message" node becomes root node of the RDF model. "Decode" method defined in RDFEncoderDecoder class looks the node with "message_context" property and understands the type of the ObjectModel. Then method starts to decode from main node.

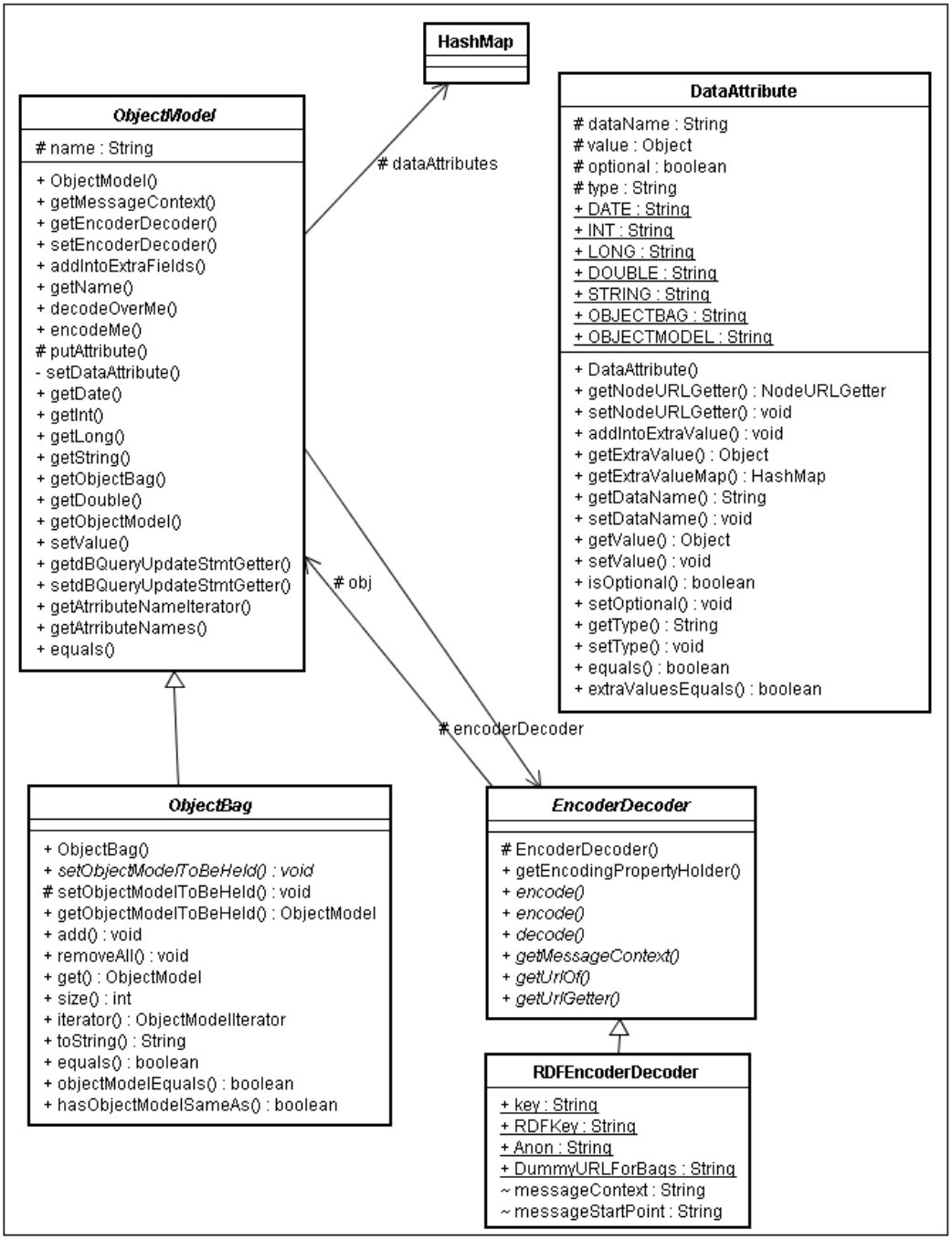

Figure 5.6 SADE data structure of messages.

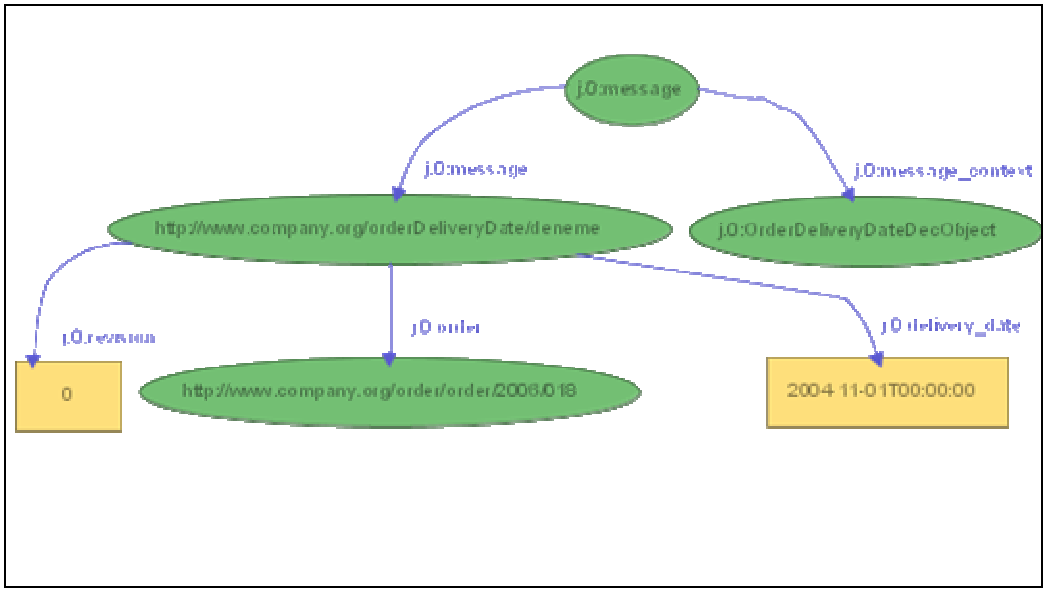

Figure 5.7 A SADE message in RDF format

CHAPTER SIX

PRODUCTION PLANNING APPLICATION

In this chapter, workflows about production planning are going to be implemented by SADE framework. The system of an organization in sector HVAC (heating, ventilation and air-conditioning) is examined for this study. Fan coil department of the organization uses a simple program working on MS Access. (Fan Coil is an indoor component of a heat pump system, used in place of a furnace, to provide additional heating on cold days when the heat pump does not provide adequate heating.) This simple desktop program is used for production planning. Planning is based on four parameters.

- Received fan coil orders from sale regions,
- Materials to produce the fan coil product, and their availability,
- The production time of products,
- Available week to produce are considered when planning facilities are realized.

Sale regions are in diverse cities in Turkey. Offices in Istanbul, Ankara, Izmir, Adana, and Antalya take the customers' orders and pass these orders to the factory in Izmir. After several communications among workers from four departments; production planning, material planning, designing, purchasing, production planner gives delivery date to sale region. All workers have desktop programs like MS Excell and MS Access to follow their duties. The organization has an ERP software but the software is not used for daily works. Accountant department is fully get benefit of the software. Other departments input data because they ought to. They do not get any output of the software because they don't know the abilities of software and the software do not meet their needs with the current configuration. Workers like to do their works with desktop programs. This concludes a diverse programs run for one system. The workers request services from each other by telephones, emails and paper forms via fax. This type of communication causes a disorder. It is concluded to

build a software system that gathers these communications and operation tasks. As to this study, the solution is using SADE framework.

Following sections in this chapter indicates the processes which are designed in SADE framework for fan coil department. These processes are running processes in SADE framework. First and the biggest process defined in this chapter is Fan Coil Order Declaration Process. This process and the application of it are described in detail in section 6.1. The others, Fan coil Order Cancellation Process, Fan coil Product Number Query Process and Reordering Materials Process are briefly described in this chapter.

6.1 Fan Coil Order Declaration Process

Fan coil order declaration workflow can be examined in figure 6.1. A production planner in Izmir collects fan coil orders and makes his/her plan and declares delivery dates of orders. Basically, production planner first looks for the first available production date for the received order after calculating the total production time of the order. Then production planner asks the material resource planner whether if there will be enough material to produce the products in the production day. If the answer is positive, the delivery date is declared to sale region. When material resource planner receives an order with a production date, he/she asks designer of the department which materials will be used for the products in the order. Material resource planner gets the material list of the order and checks the availability of these materials in the production week. If there will be enough material in the week, a positive answer is sent to production planner. If there won't be enough material, he/she asks to purchasing department when these materials can be received. Material resource planner gives this information to the production planner. Production planner decides when the order is put into production and declares the delivery date of the order. At this stage, another process, material order plan process, is followed. Fan coil order declaration workflow can be examined in figure 6.1.

According to the method described in section 4.5, this process may be analyzed like following sub sections.

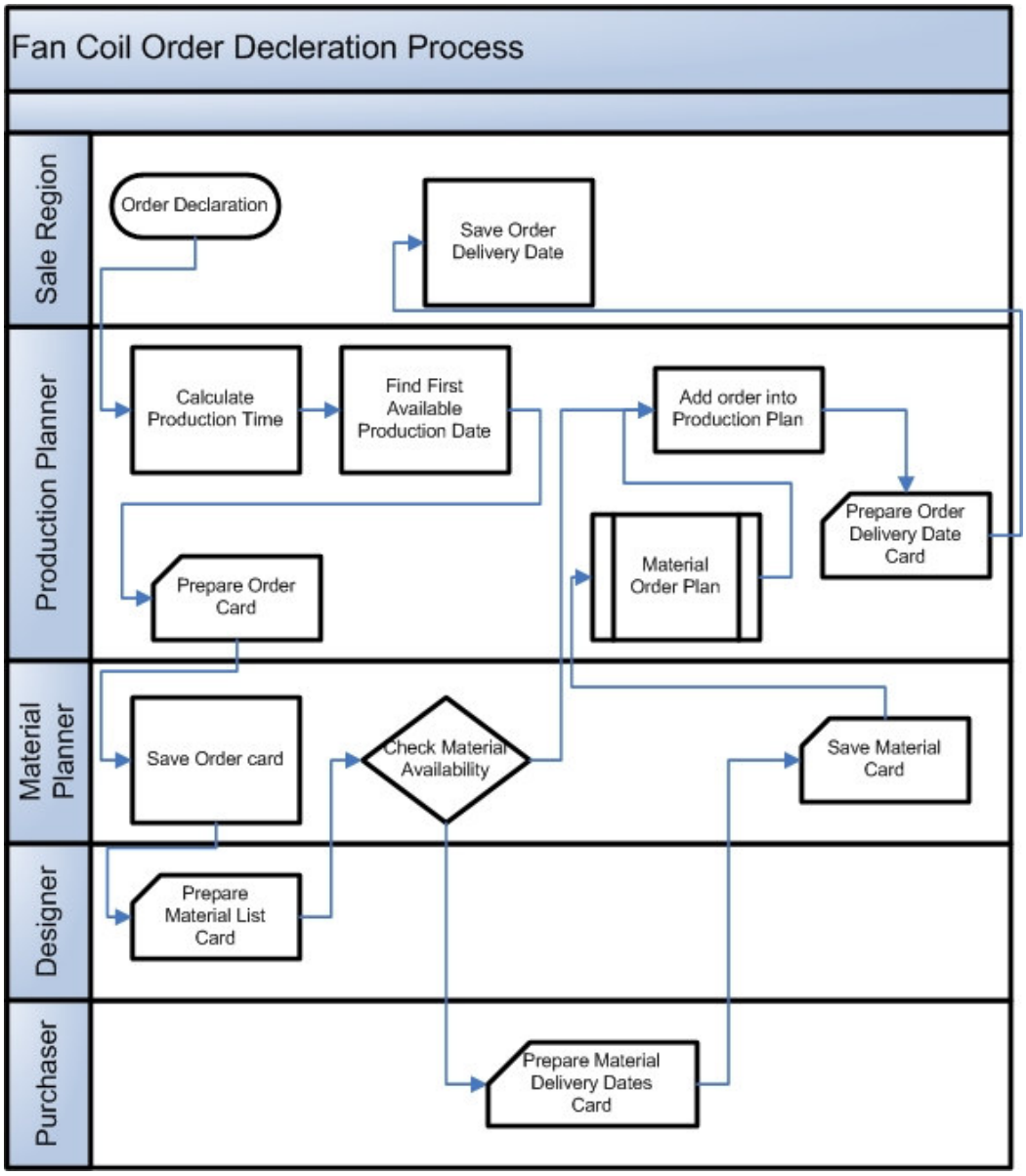

Figure 6.1 Fan coil order declaration process.

6.1.1 Step 1, Defining Process

The subject of the process is fan coil order declaration. The order declaration is for putting the order, which is sent from a sale region, in production plan and determining a delivery date.

Orders' containing may change in time. Sometimes the number of fan coils changes, sometimes the fan coil configuration changes. That's why the declarations of orders must have a revision number to distinguish new order from revision of an order. New orders have revision number 0. The others have numbers greater than 0.

Figure 6.1 can be described in this step. The figure has been described earlier in this section so it is not preferred to be mentioned again.

6.1.2 Step 2, Defining Actors

The actors are sale region, production planner, material planner, designer, and purchaser. According to the actors, there will be a core agent for every actor. Besides that, there will be gui agents for sale region, production planner and material planner.

6.1.3 Step 3, Duties of Each Agent

Sale region gui agent gets the order declaration from its user. Sale region core agent passes the order message to production planner core agent. Sale region user declares a desired delivery date for the order. If the returned delivery date is after desired delivery date, the core agent of sale region puts a warning message into user's inbox of the gui agent.

Production planner core agent receives the orders and finds first available date for the order. Then it passes the order to the core agent of material planner with the possible production date. If the answer of the material planner is positive, the production planner core agent returns the delivery date for the order to the sale region. If the answer is negative, the received message will have a material list that must be ordered to suppliers. This message and the order message is put into user's inbox of gui agent by core agent. The decision of the production date and delivery date is left to the user of production planner gui agent.

Core agent of material planner receives the order and first asks to the designer which materials need for the production of the order. After it gets the list of material,

it checks the storeroom for their availability at the time the production will occur. If there is enough material, it returns a positive message to production planner. Otherwise, it sends a message, which contains the material list that is needed, to purchaser core agent. Core agent of purchaser responds when the materials will be received after the order of materials. This respond is sent to production planner as it is.

Core agent of designer responds only the material list queries in this process. Core agent of purchaser responds only the material list to be ordered.

6.1.4 Step 4, Relationships Among Actors

Sale region agent asks production planner the delivery date of the order. Production planner asks the material planner if there will be enough material in the proposed production date. Material planner asks the designer which materials are needed in production and asks purchaser the materials' lead times after material orders. Figure 6.2 summarizes the relationships among actors in the sequence diagram. JADE has a debugging tool called "Sniffer" which figures the messages sent among agents. Figure 6.3 shows the messages sent during the flow of fan coil order declaration process in the application by the help of "Sniffer" tool. The agents shown in figure are core agents of domains. These two figures are very similar. This also shows that agent systems have great capability of the simulation of real systems.

The first "Request" line in figure 6.3, shows the message sent from gui agent of sale region to its core agent. The user of sale region agent starts the flow when he/she enters an order into system. Couples of "Request- Inform" lines show the messages for asking the agent addresses to DF (Directory Facilitator).

6.1.5 Step-5, Defining Data Structure of Messages Delivered Among Agents

The contents of messages sent among agents are decided to be encoded in RDF. The graph forms of RDF messages are figured and the explanations of the message are at the below of the figures.

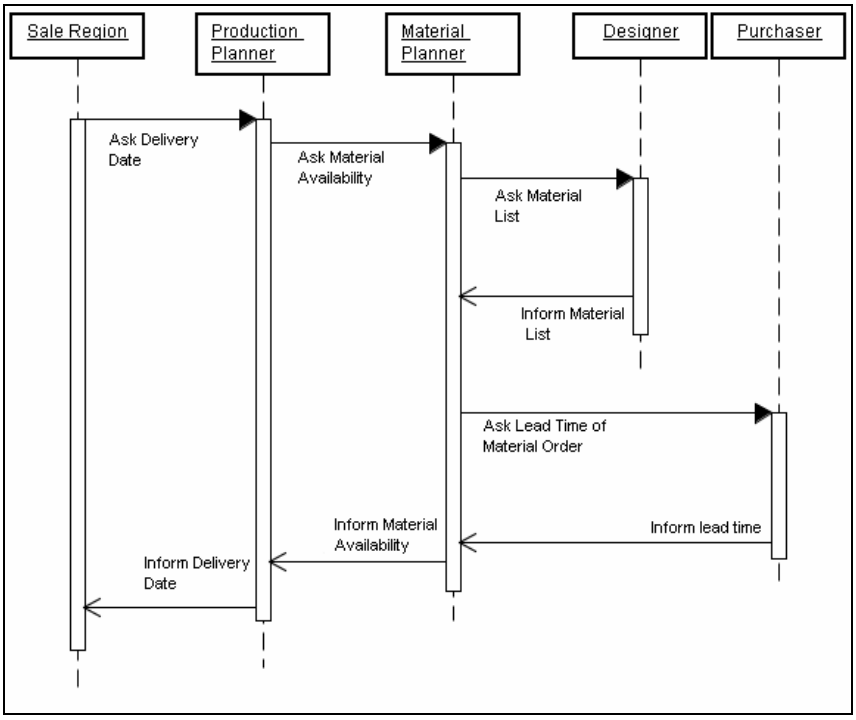

Figure 6.2 Sequence diagram of fon coil order declaration process is running.

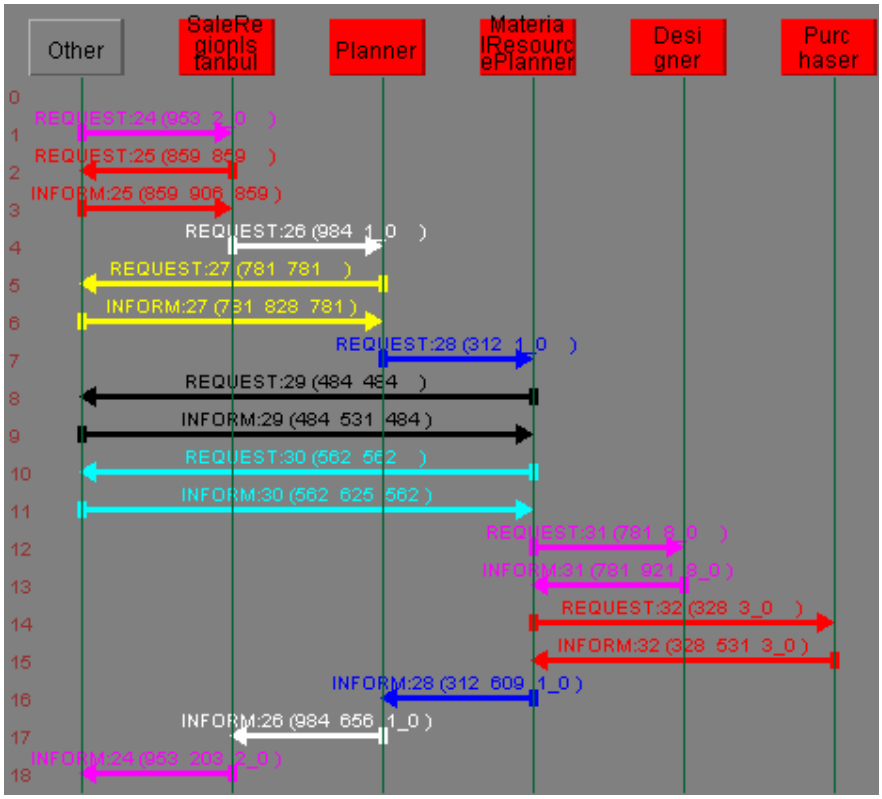

Figure 6.3 JADE messages while fon coil order declaration process is running.

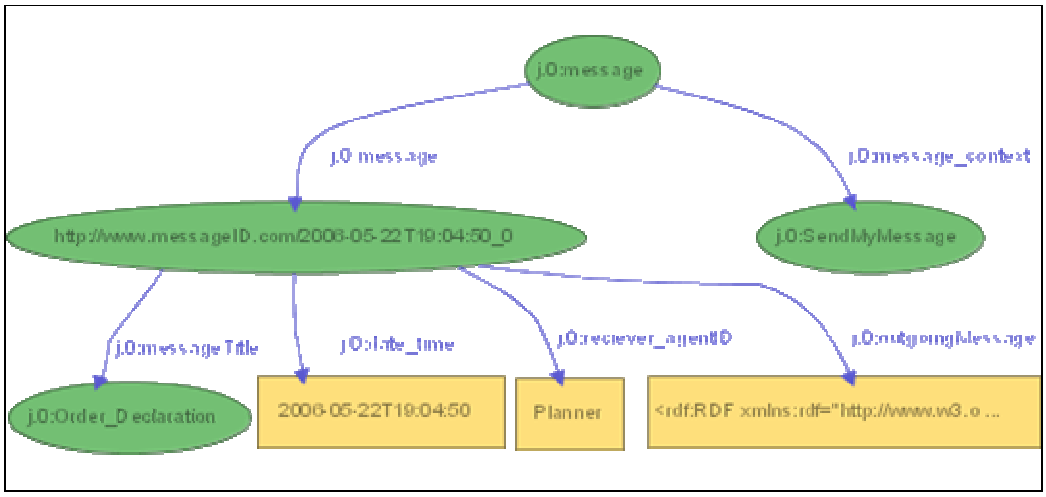

Figure 6.4 A request message which is sent from gui agent of sale region to its core agent.

The type of messages can be identified by the nodes which are pointed by "message_context" labeled arcs. In other words, objects of "message_context" predicates, define the message types. As to this description, 6.4 shows the message (or RDF format of ObjectModel object) of "SendMyMessage". This type of message is used between a gui agent and a core agent. Core agent gets the message. Core agent sends the literal object which is pointed by "outgoingMessage" predicate to agent which is determined by "receiver_AgentID" predicate. The others nodes give extra information about message that can be used for logging the messages.

Figure 6.5 shows the message sent from core agent of sale region to core agent of production planner. This message and the literal node specified with "outgoingMessage" predicate in figure 6.4 are literally same. Figure 6.5 depicts the object Order_Declaration in RDF format. Order declaration number is specified in the node which is pointed by "message" labeled arc in the figure. The orders usually are revised because customers change their ideas occasionally. That's why a revision number is placed in the message. "revision_no" predicate specifies the revision number. Revision number 0 means the first declaration of the order. Customers usually want to know when they will receive the fan coils because fan coils must be placed in the building in a certain stage of construction. That's why sale region gives

a delivery date before asking to production planner when the sale region is trying to sell the fan coil product. Because of that, desired delivery date is specified by

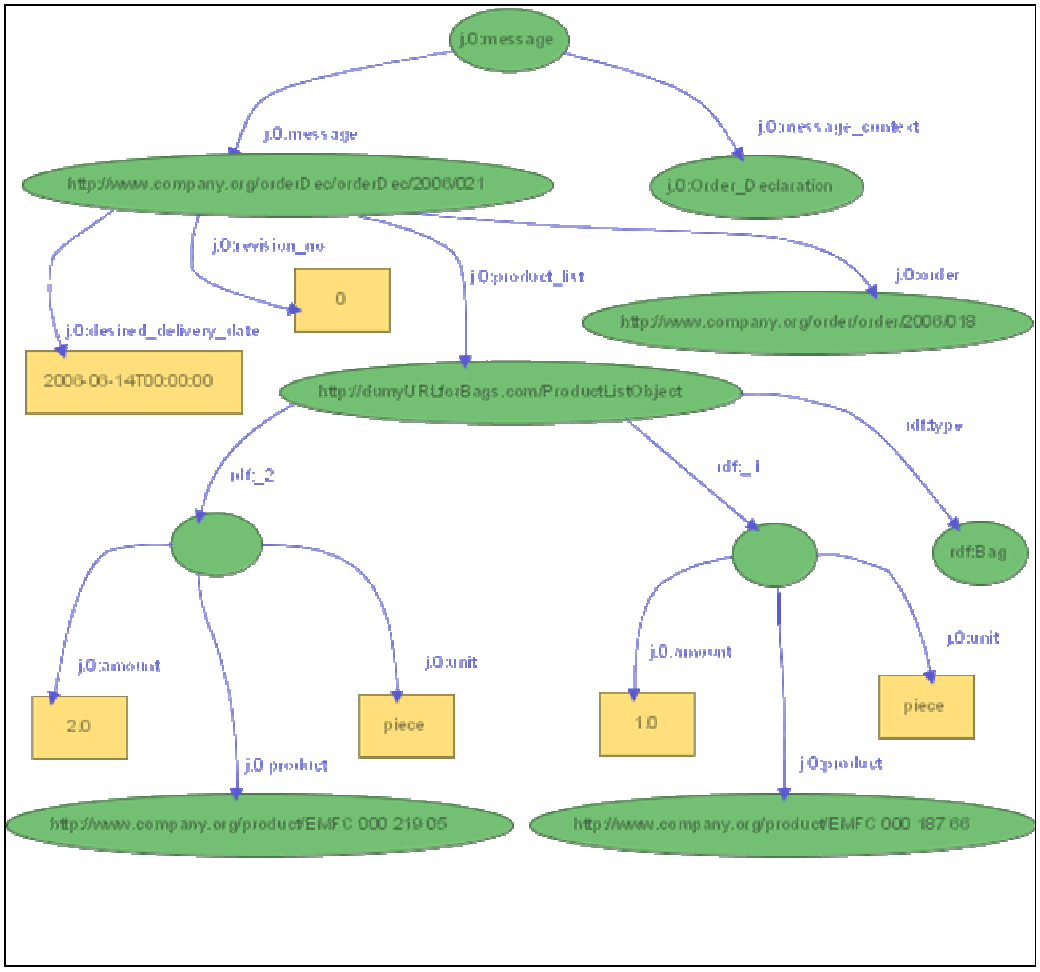

Figure 6.5 The message sent from core agent of sale region to core agent of production planner for delivery date request with order declaration details.

“desired_delivery_date” predicate in the message. Of course, there will be a product list in order declaration. It is denoted by “product_list” predicate in the graph. Product list node is actually type of “Bag” (in the sense of RDF terminology). The bag holds product numbers, amount of every product and the unit of the amounts. In this example, fan coils can be ordered each by each. In figure 6.5, two sorts of fan coil are ordered. One of them is ordered as a two pieces and the other one is ordered as a one piece. You may consider the benefits of using RDF in this example. The

products are defined as resources and they have their own URIs. The resources of firm have unique identifiers. This gives a great opportunity to build a knowledge based system.

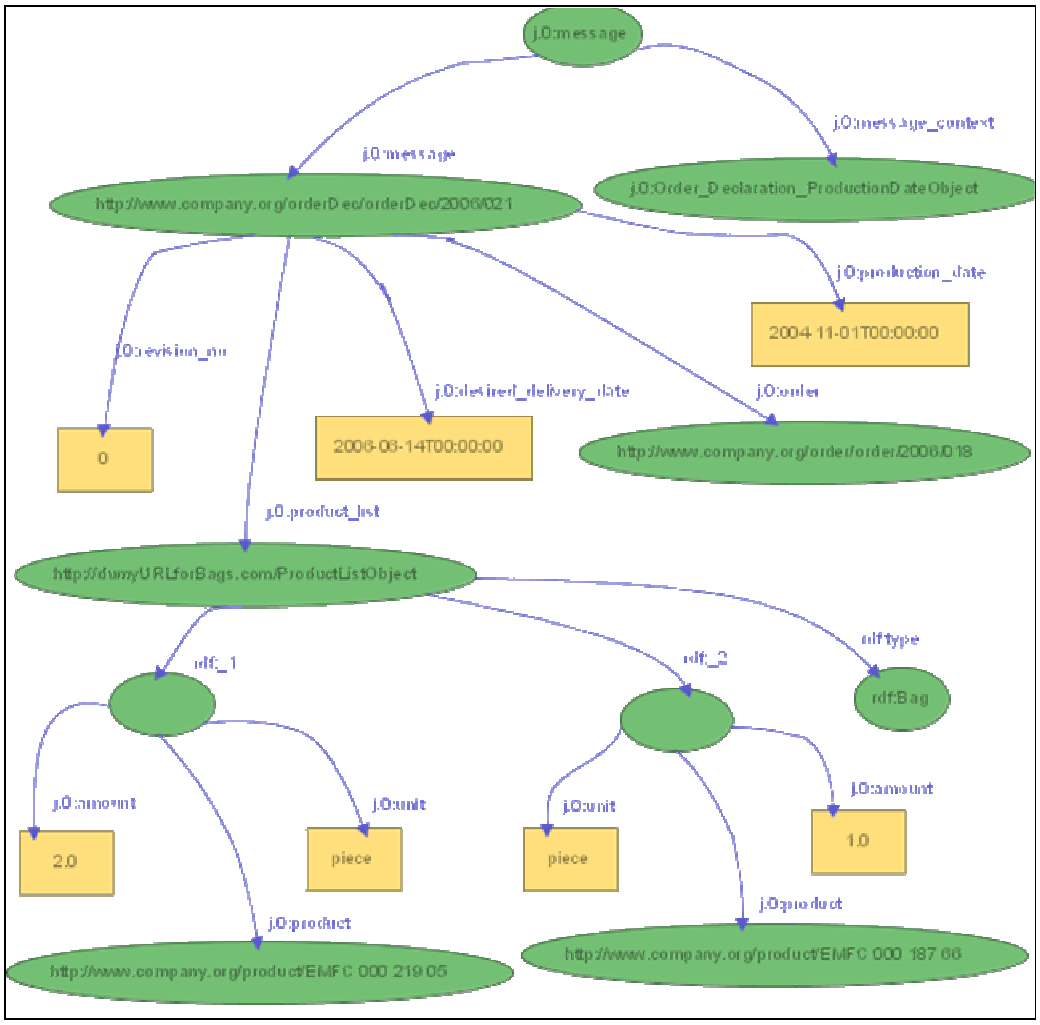

Figure 6.6 The message sent from core agent of production planner to core agent of material planner.

The message shown in figure 6.6 is sent from core agent of production planner to material planner. The difference between the messages in figure 6.5 and 6.6 is the existence of “production_date” predicate. All other properties are same. The material planner checks for the material availability in the proposed production date. This message is also sent to core agent of designer to get the material list. It is decided that another message is not needed to be designed for this conversation.

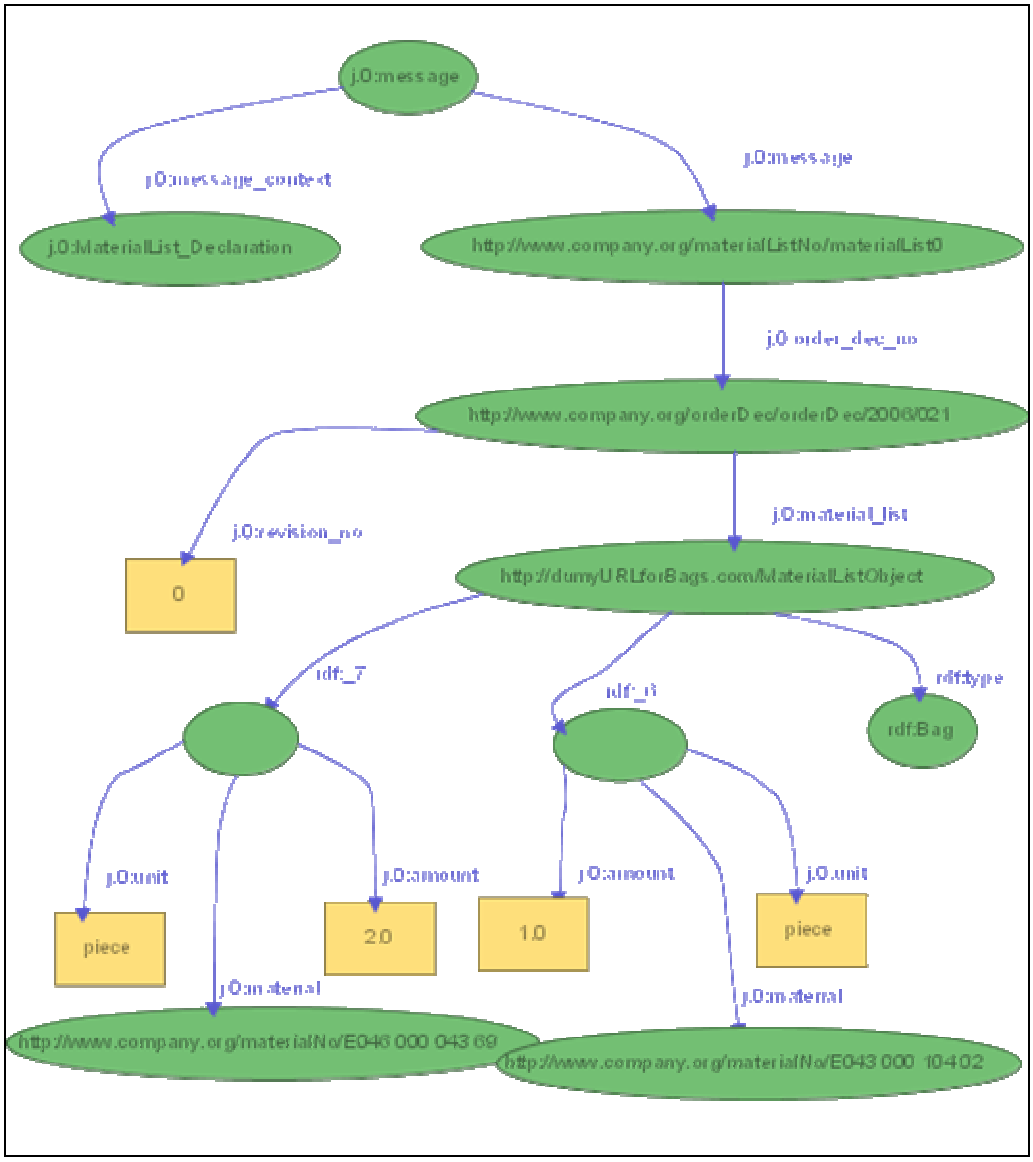

Figure 6.7 The message sent from core agent of designer to core agent of material planner as response to the message in figure 6.6.

Figure 6.7 depicts the message sent from core agent of designer to core agent of material planner as response to the request of the material list of the order. Material list is specified with “material_list” in the model. The node of order declaration number is chosen as the subject of the predicate. When the models in figure 6.7 and 6.6 are joined together, the products and materials of the order can be gathered in one RDF model. This will provide us to infer about what is going on the system.

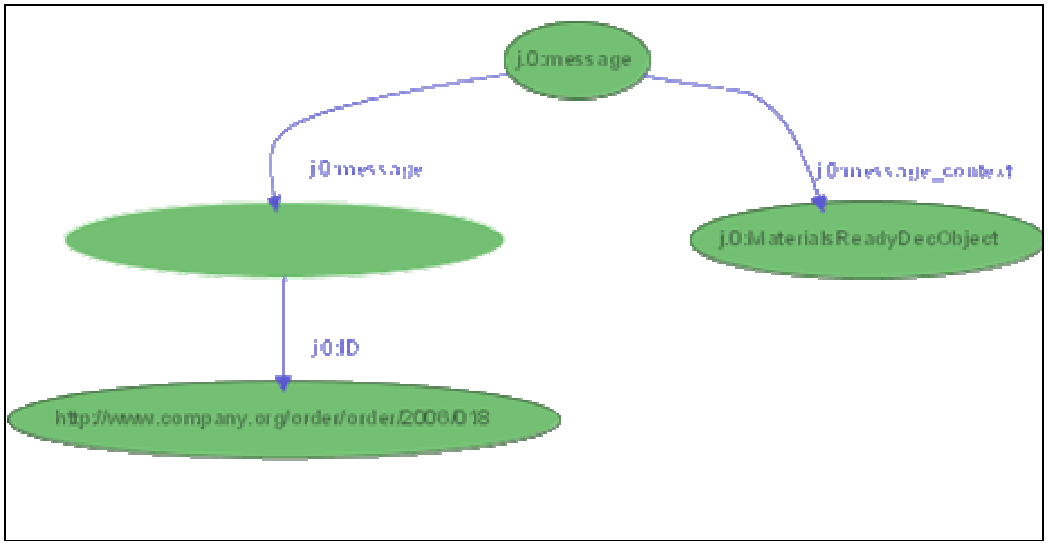

Figure 6.8 The message sent from core agent of material planner to core agent of production planner for declaring that the materials are going to be available at the production date.

Figure 6.8 shows the message from core agent of material planner to core agent of production planner as response to question about the availability of production materials in production day. This message means that there will be enough materials to produce the fan coils of the order. The message contains the order number of the order to avoid of misunderstanding between agents.

Figure 6.9 shows the message from core agent of production planner to core agent of sale region as response to request for delivery date of the order sent from sale region. This message defines the delivery date via “delivery_date” predicate.

Figure 6.10 shows the message sent from core agent of material planner to core agent of purchaser for requesting the lead times of material order. The material order is also associated with the order declaration for the opportunity of building a knowledge system easily.

Figure 6.11 shows the message sent from core agent of purchaser to core agent of material planner as to response of the request of lead times of the material order.

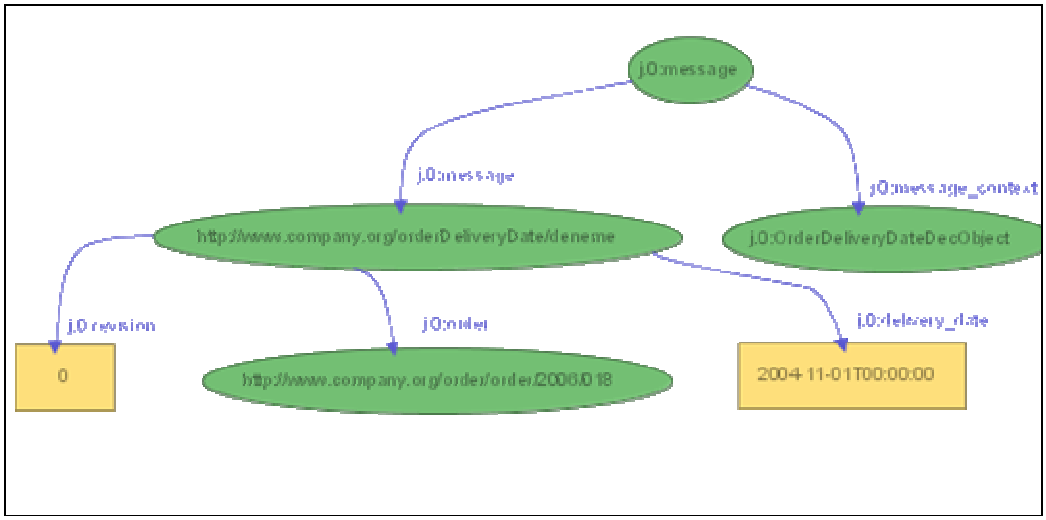

Figure 6.9 The message sent from core agent of production planner to core agent of sale region for declaring that the delivery date of the order.

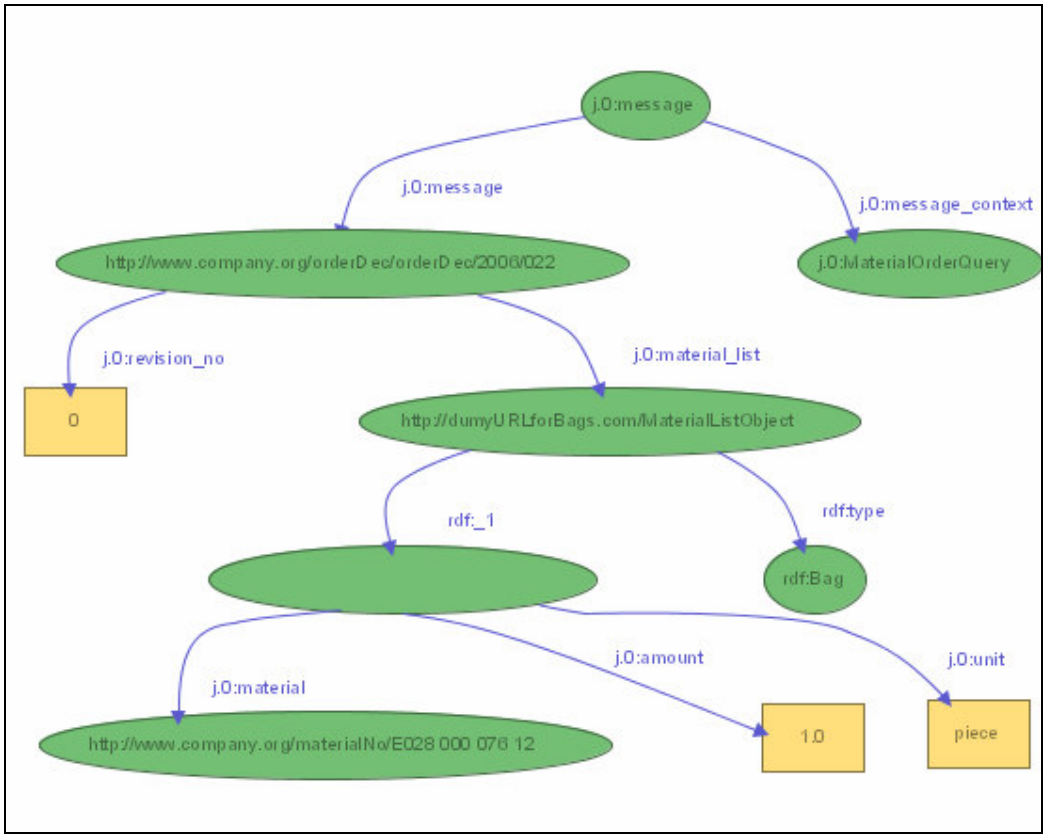

Figure 6.10 The message sent from core agent of material planner to core agent of purchaser for requesting the lead times of material order.

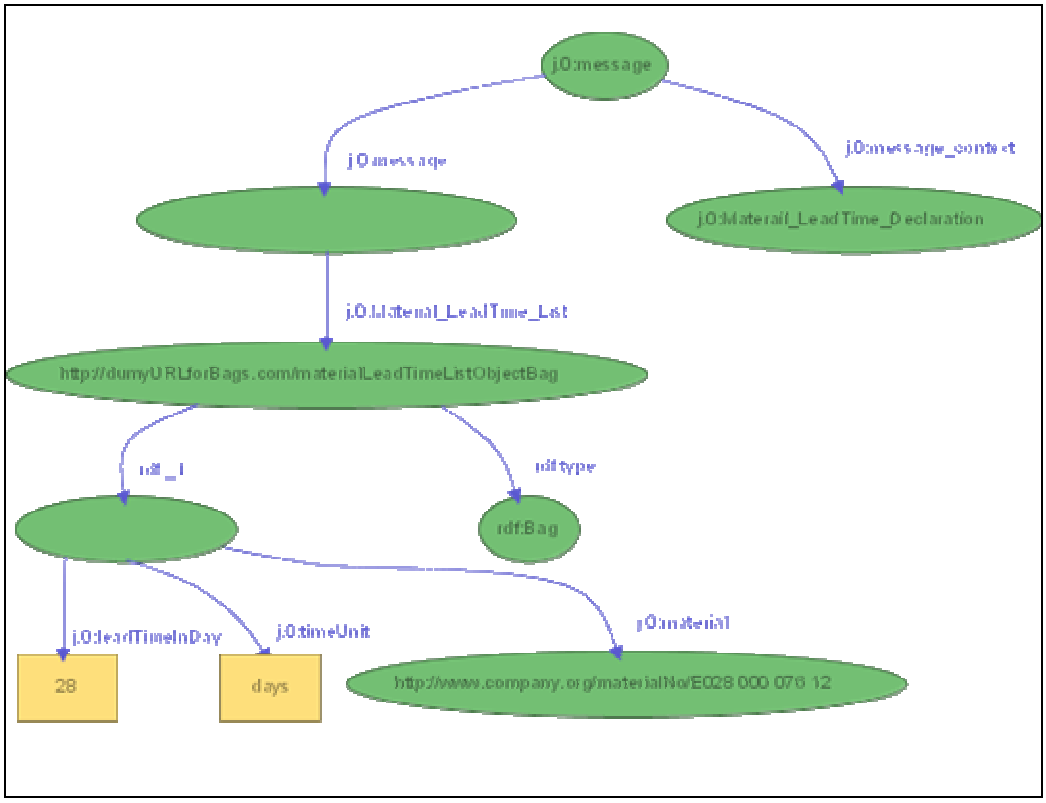

Figure 6.11 The message sent from core agent of purchaser to core agent of material planner for declaring the lead times of material orders.

6.1.6 Step 6, Goals of Agents

Core agent of sale region checks whether the desired date of order delivery is got or not. Core agent of production planner aims to give earliest delivery date when an order is received. Core agent of material planner checks the availabilities of materials in a time interval. The other core agents of designer and purchaser do not have goals in this workflow. They only respond the requests from material planner.

6.1.7 Step 7, Defining Computing Step Taken In Agents:

SADE framework has a simple tool for ready Step Groups to ease the debug operations during development of Step Groups. The tool gets a Step Group and outputs a simple RDF model represents the Step Group. This tool will be used in this section. The outcome of the tool will be viewed as graphs by the help of Isaviz RDF editor v2.1.

Nodes in different types are figured differently in figure 4.2. However, the tool used in the scope of this study is not so definitive. It is a prototype tool. It only creates an RDF model that may present the names of the steps in flow. The types of the step nodes may be added and a special program that visualizes different nodes in different styles can be developed. Nevertheless, the current RDF model can help to understand the steps taken in agents.

Because the figures of some graphs in this section are too large to be viewed in one page, the graphs are split into several figures. And also the RDF nodes are numbered to make more understandable. The numbers in figures are not part of the RDF model.

Figure 6.12 shows the first part of the steps taken in the core agent of sale region when an order declaration is received from gui agent of sale region. The first nodes of every step groups are the nodes that hold the step group name, data store and some other initial values of the step group. These nodes are in type of StepGroup object. So that the name of the step group in figure 6.12 is Workflow_OrderDeclaration_SaleRegionCore. The first step after StepGroup object is StarterResumerStep object. After initializations of step group, the incoming message is stored into data store by node 3. The real message is wrapped with another object, called SendingObjectModel, used between gui agents and core agents. The wrapper object contains the destination of the message and some other metadata about message. The core agent can understand what should be done with the message by the wrapper object.

Besides that, in this case, the core agent of sale region has some duties other than bypassing message. Because of that, the order declaration object should be extracted and put into data store in node 4. Afterwards the order is inserted into database of sale region. Node 6, sends the incoming message to planner as described in wrapper message. Because the execution of step group pauses, SADE looks for any message to be return to gui agent of sale region. In this stage, there is nothing to return. Thus,

this is mentioned in node 7. Node 8 make the flow pause to wait for the response from production planner. Node 9 is executed when the message has come from

Figure 6.12 The steps taken in core agent of sale region, part one.

production planner. This node checks the incoming message type and chooses the proper next step. In this flow, the paths to be followed for the messages in types of refuse, agree and inform. Node 10, only, prints into command prompt a message that

says that a refuse message has come. Then the flow is ended. Node 11, only, prints into command prompt a message that says that a agree message has come. Then the flow is ended. When the inform message comes, node 12 put the message into data store. If newly incoming message has no syntactic error the node 13 is executed. Node 13 is pointed by the arc which is labeled as nextStep_UNDERSTOOD. Otherwise, the nextStep_NOTUNDERSTOOD arc should be fallowed. In this flow there is no arc labeled like that. Thus, the execution of flow will stop immediately, if the incoming message has syntactic error. Node 13 puts the incoming message from production planner into inbox of gui agent of sale region. Node 14 sends the incoming message to gui agent of sale region. If any message is not sent to gui agent, the time out procedure will run. Time out procedure puts the sent message and a related timeout message into message inbox of gui agent. If any message is not sent to gui agent from core agent of sale region, there will be a timeout message and delivery date message for the same order declaration message.

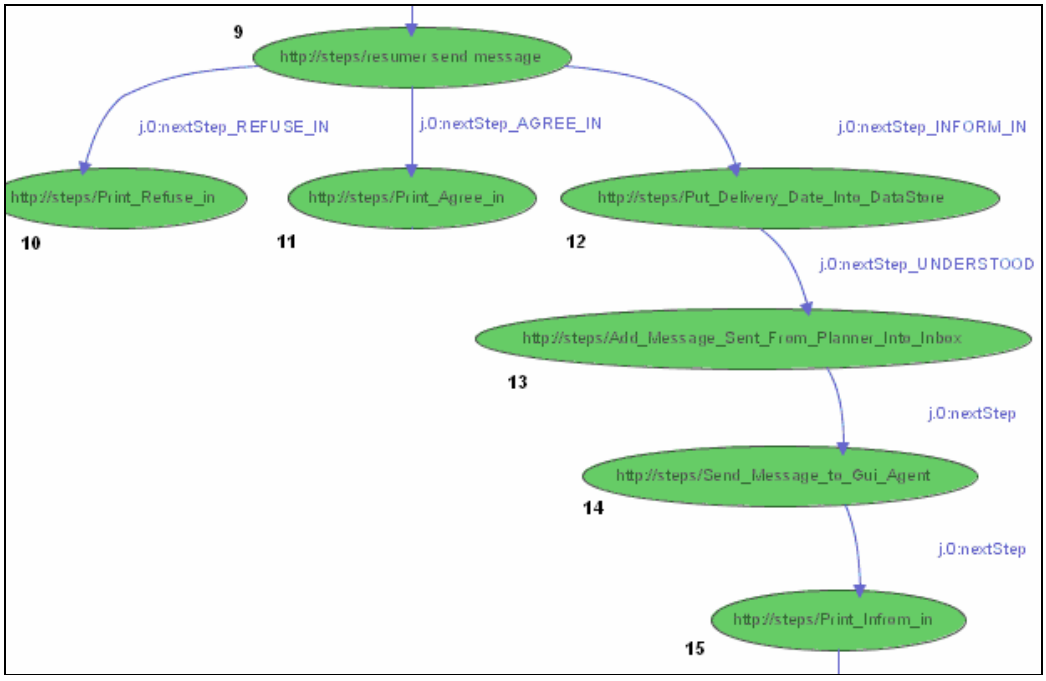

Figure 6.13 The steps taken in core agent of sale region, part two.

Node 15, only, prints into command prompt a message that says that an inform message has come. Node 16 tests if the incoming message is type of message that

determines the delivery date of the order. Production planner can also send a message telling that the user of production planner agent will determine the delivery date. That's why the message is checked. If the test is failed, the execution of flow is stopped because there is no specified node with nextFalseStep arc in the graph. Otherwise, the delivery date of the order is inserted into database of the sale region in node 17. Node 18 gets the delivery date and the desired delivery date from the database into data store. Node 19 tests if delivery date is later than desired delivery date. Node 20 creates a related message that tells desired delivery date is later than the desired date. Node 21 inserts this message into inbox of sale region. The execution is ended after node 21 is executed.

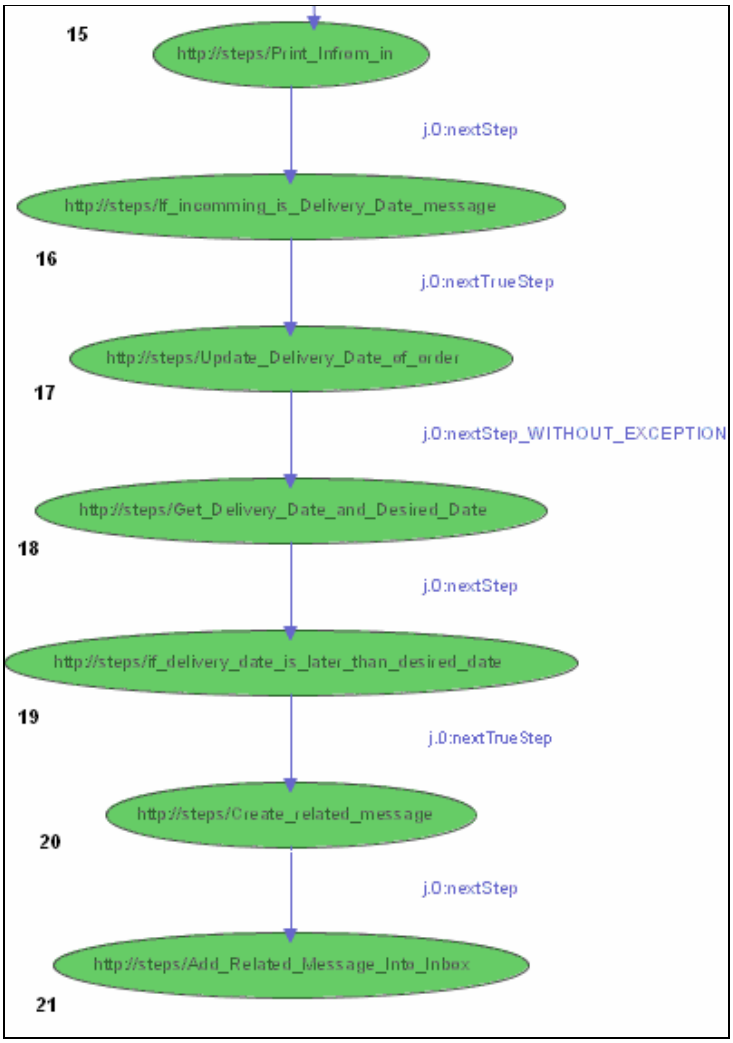

Figure 6.14 The steps taken in core agent of sale region, part three.

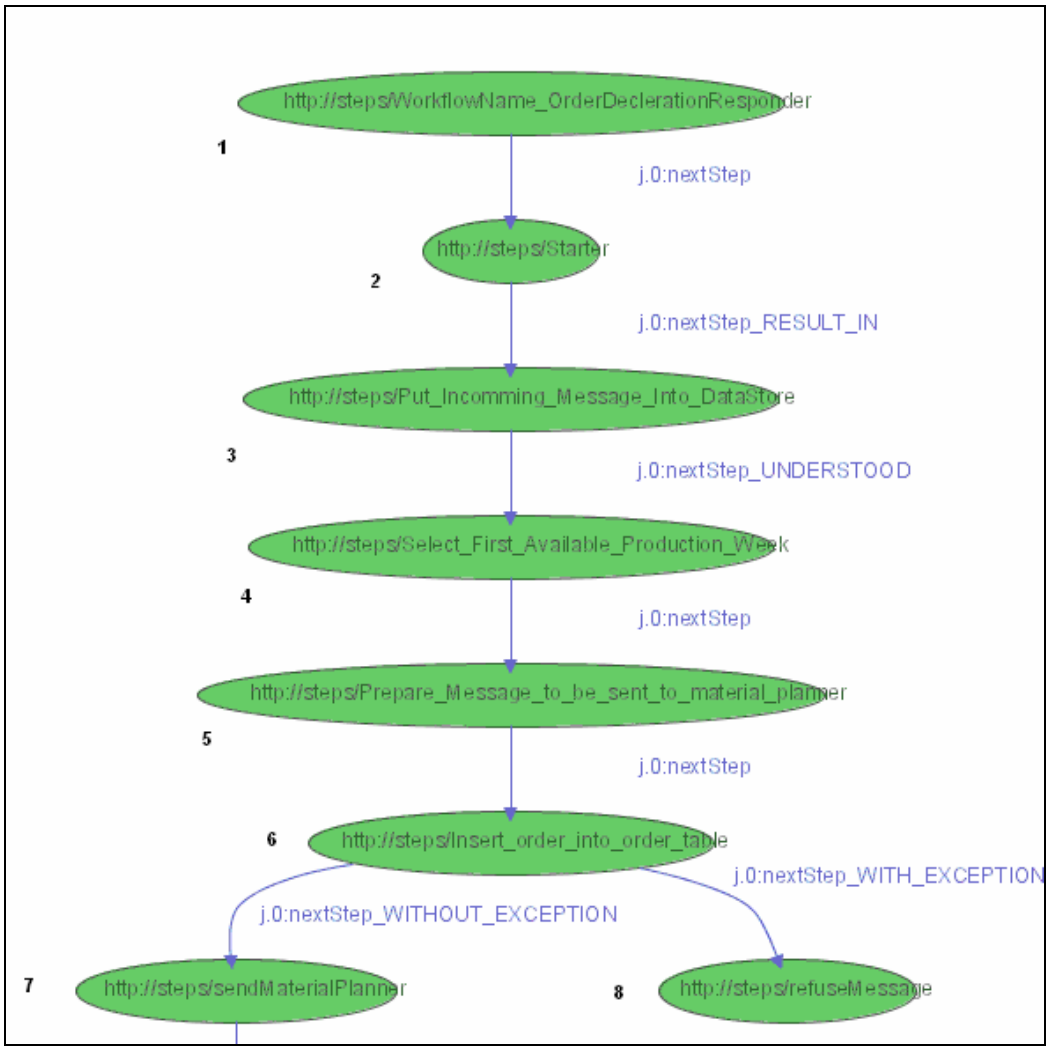

Figure 6.15 The steps taken in core agent of production planner, part one.

Figure 6.15 , 6.16 and 6.17 show the steps taken in core agent of production planner. Node 1 denotes the name of the step group which is WorkflowName_OrederDeclarationResponder. Node 2 starts the flow. Node 3 inserts the incoming message, which is declaration of fan coil order, into data store. Node 4 queries the database for the first available production week and inserts the date into data store. Node 5 gets the production date from data store and constructs the message to be sent to material planner. Node 6 inserts the order into database of production planner. If the insertion returns with an error, node 8 is executed. Node 8

sends sale region a refuse message. If the insertion is completed without error, the message prepared in node 5 is sent to core agent of material planner in node 7. Node 9 pauses the execution of flow to wait for the response from material planner. Node 10 resumes the execution of the flow. Node 11 inserts the incoming message from material planner into data store. Core agent of material planner can send a message that tells that there are enough materials at production week or a message which contains a material list that must be ordered. Node 12 tests if the incoming message is a list of material that should be ordered.

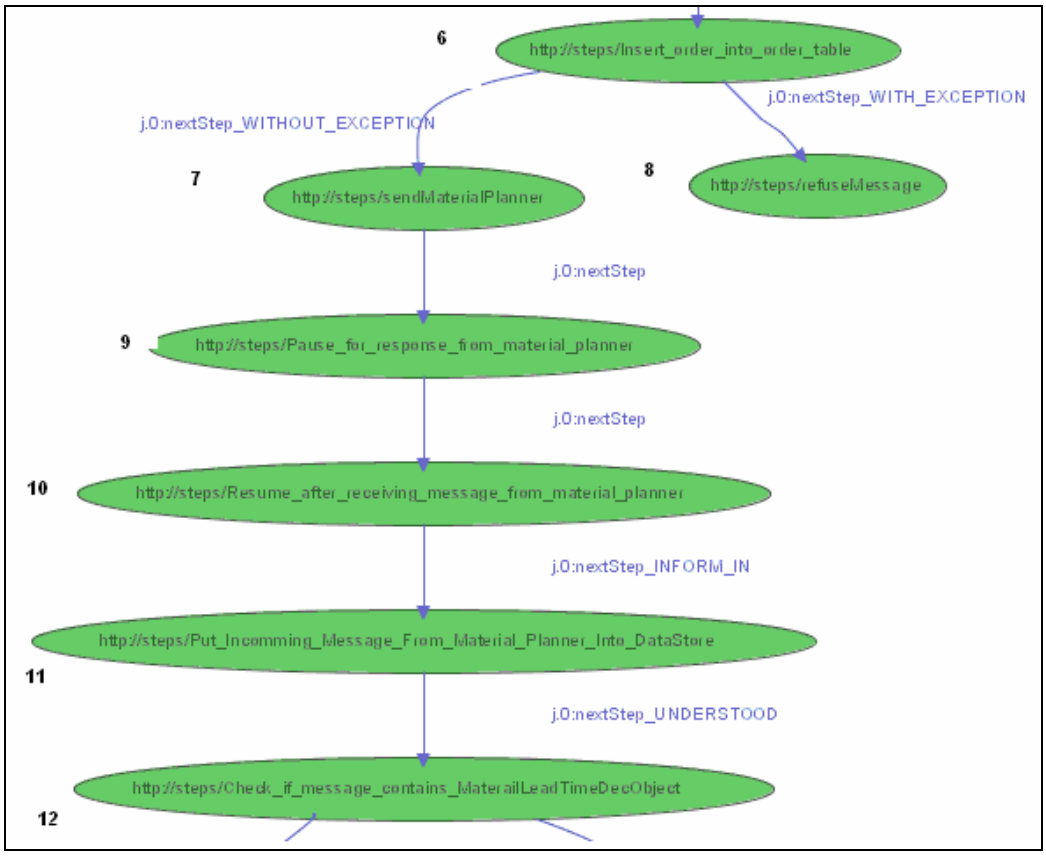

Figure 6.16 The steps taken in core agent of production planner, part two.

Node 13 tests if the incoming message is message that tells the materials will be ready in production date. If test is completed successfully, node 14 inserts the order into production plan. Node 15 constructs the returned message to sale region. Node 16, 17 and 18 sets the fields of returned message like order declaration number, delivery date. Node 19 finally sends the delivery date message.

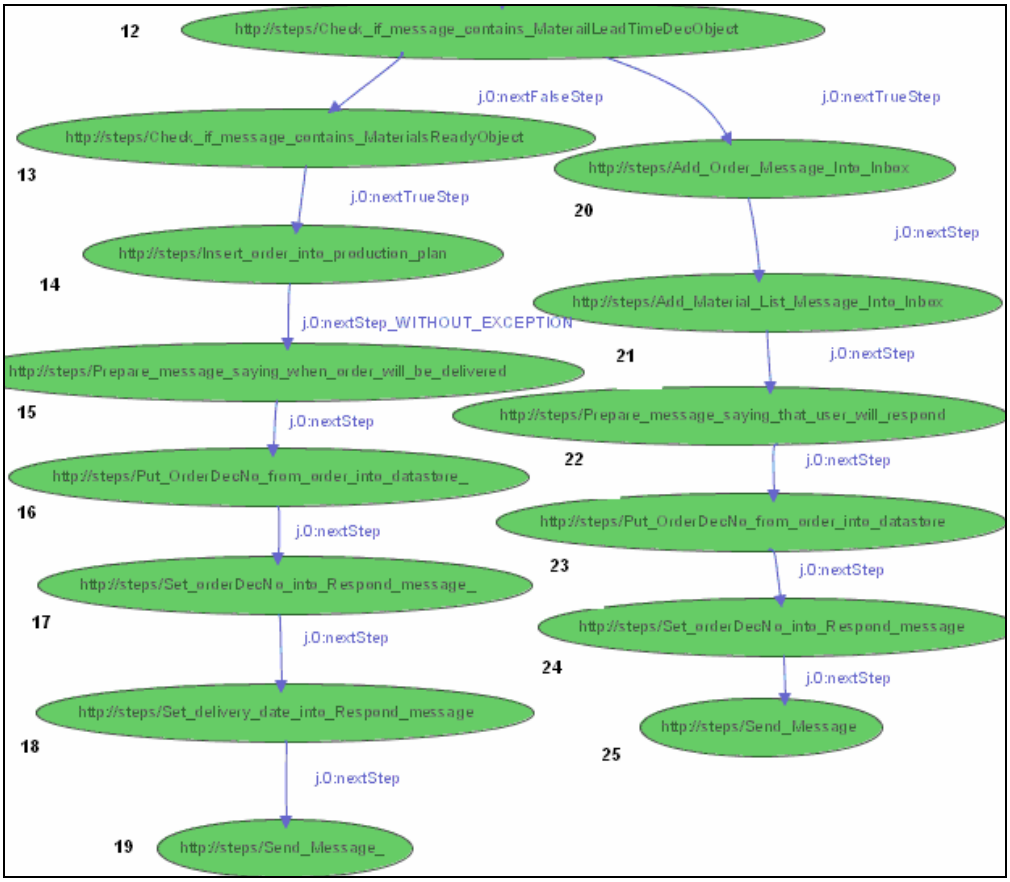

Figure 6.17 The steps taken in core agent of production planner, part three.

Node 20 is executed when the material list to be ordered comes. Node 20 inserts order declaration message into inbox of production planner. Node 21 inserts the message which contains material list to be ordered. Node 22 prepares a message telling that the order delivery date will be decided by user of the production planner agent. This is needed because the sale region waits for an answer. If no answer is sent, the sale region may wait for ever if a time out is not set. Node 23 and 24 sets the fields of return message.

Figure 6.18, 6.19 and 6.20 show the steps taken in core agent of material planner. Node 1 denotes the name of the step group. Node 2 starts the flow. Node 3 inserts the incoming message, which is declaration of fan coil order, into data store. Node 4 sends the order declaration to the core agent of designer. Node 5 denotes that no

response message is sent in next pause of flow. Node 6 pauses the flow. Node 7 resumes the flow when the message from designer comes. If the incoming message from designer is a agree message, it is sent to production planner in node 12. If it is inform message, the material list for the production of the order sent from designer is stored in data store in node 8. Node 9 adds probable production date which is sent from production planner into material list object. This date is going to be used for the query to test whether enough material will exist in the production date. Node 10 queries the database and stores the result via material list object which also contains the probable production date.

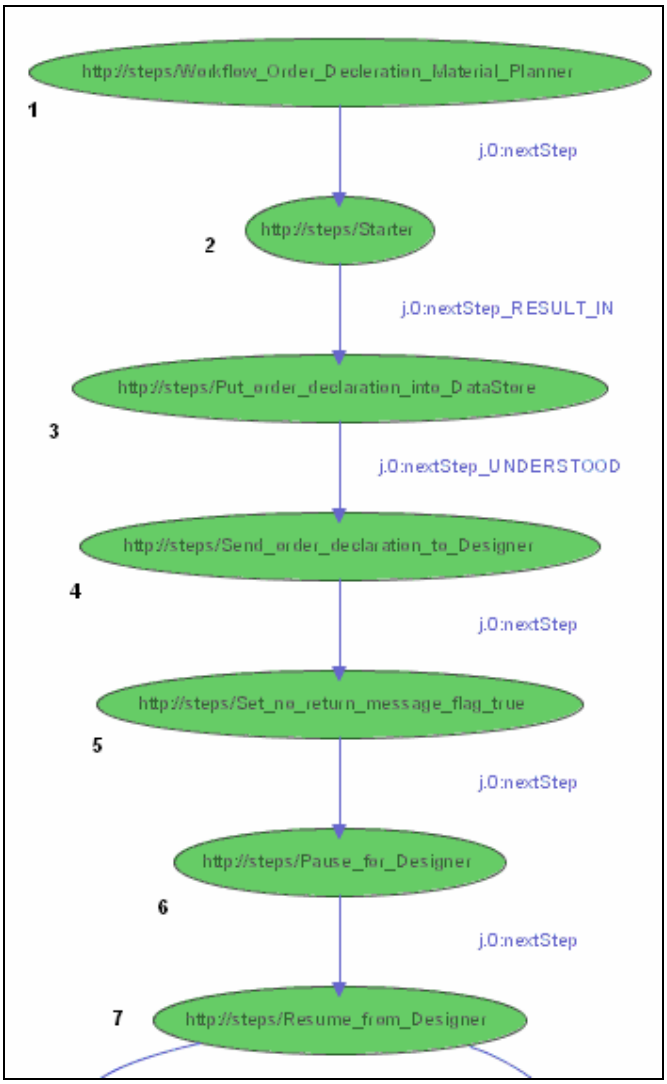

Figure 6.18 The steps taken in core agent of material planner, part one.

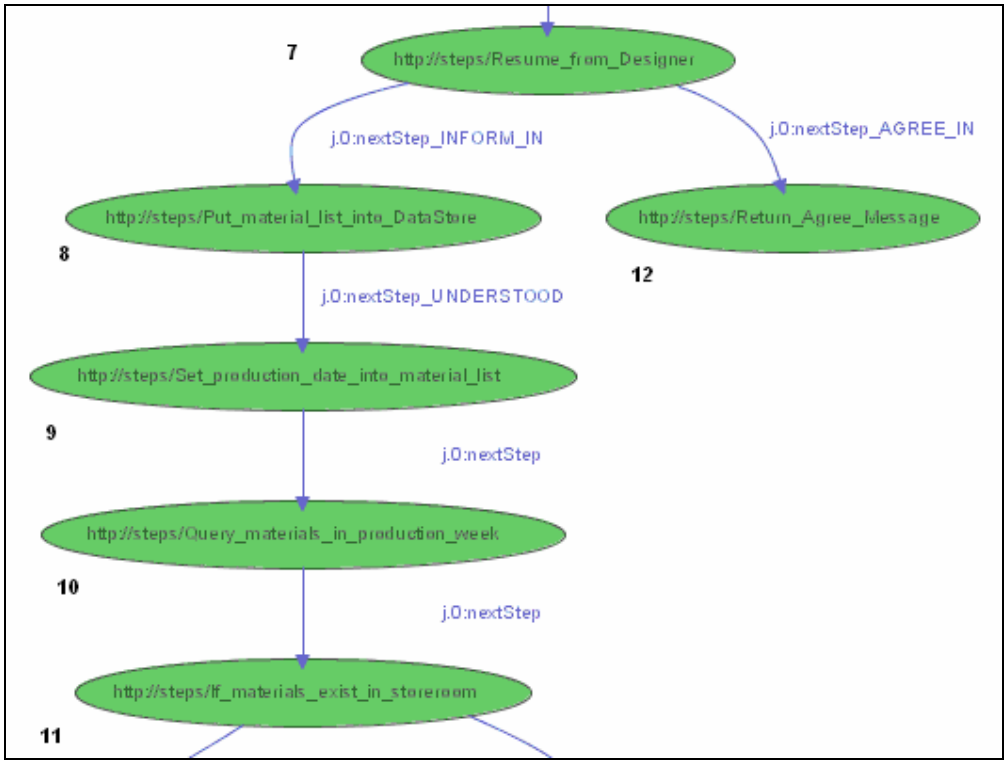

Figure 6.19 The steps taken in core agent of material planner, part two.

Node 11 tests if the result set, which is created by node 10, has any value that identifies lack of material in production date. Node 13 reserves materials via inserting proper lines into database. Node 14 constructs the positive message to be sent to production planner. Node 15 and 16 set the order declaration number field and order number field of returned message respectively. Node 17 sends the message to production planner.

Node 18 queries the database and stores the result. Node 19 constructs the message to be sent to purchaser. The message contains the materials which are needed in production. Node 20 sends the message to purchaser. Node 21 pauses the flow. Node 22 resumes the flow after the response from purchaser comes. Node 23 stores the incoming message into data store. Node 24 sends the message to production planner. Finally flow is ended after the execution of node 24.

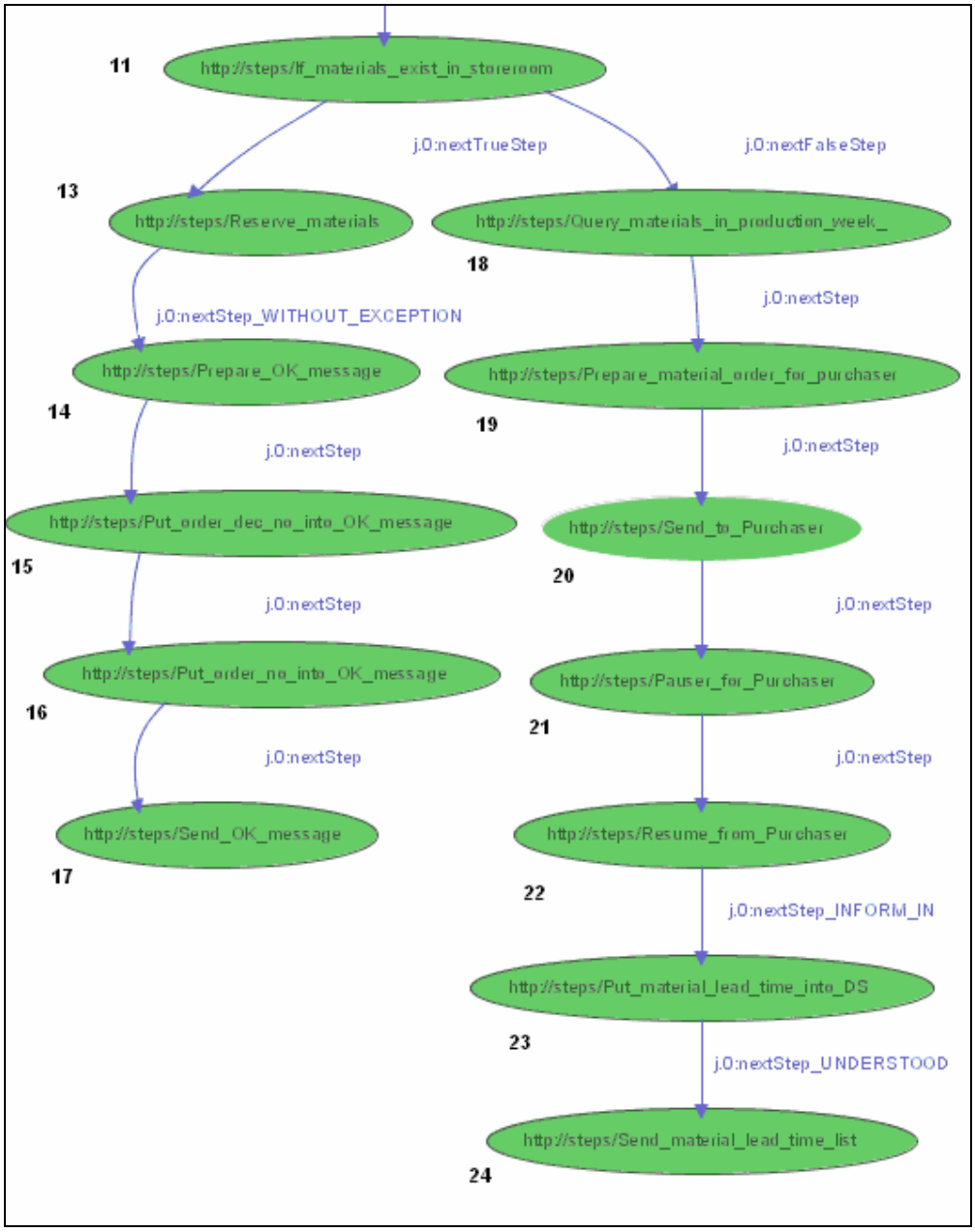

Figure 6.20 The steps taken in core agent of material planner, part three.

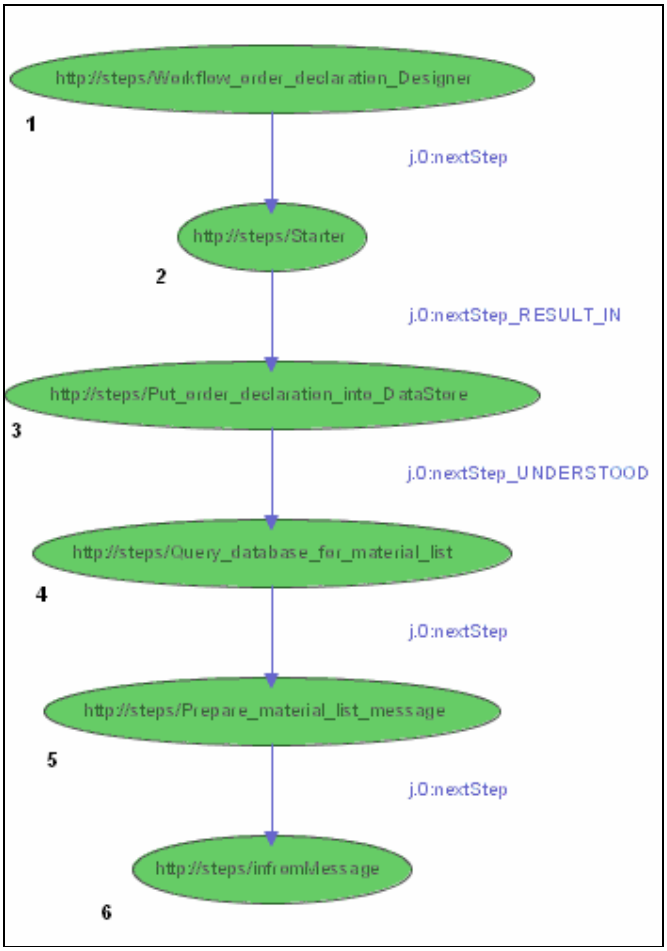

Figure 6.21 The steps taken in core agent of designer.

Figure 6.21 shows the steps taken in core agent of designer. Node 1 denotes the name of the step group. Node 2 starts the flow. Node 3 inserts the incoming message, which is declaration of fan coil order, into data store. Node 4 queries the database for material list that is needed for order and stores the result set in data stores. Node 5 constructs the message that contains the material list of order. Node 6 sends the message to material planer. The flow is ended after the execution of node 6.

Figure 6.22 shows the steps taken in core agent of designer. Node 1 denotes the name of the step group. Node 2 starts the flow. Node 3 inserts the incoming message, which is list of lacking materials, into data store. Node 4 queries the database for lead times that will pass after material orders. Node 4 also stores the result set from the

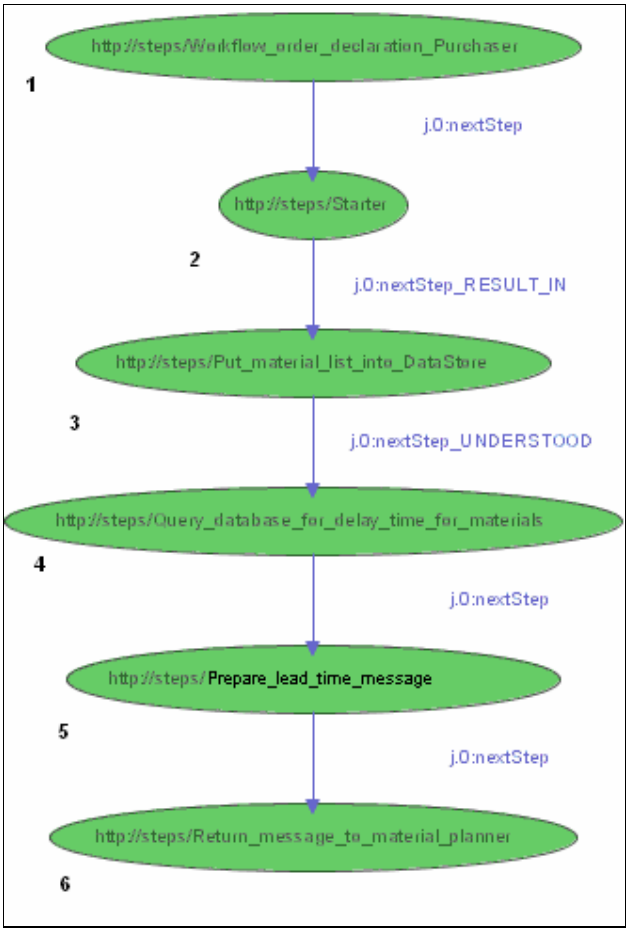

Figure 6.22 The steps taken in core agent of purchaser.

query into the data stores. Node 5 constructs the message that contains lead times. Node 6 sends the message to material planer. The flow is ended after the execution of node 6.

6.2 Fan Coil Order Cancellation Process

Order cancellation process is very simple. Sale region declares the order cancellation to production planner. The planner discards the order from production plan and sends the order cancellation information to the material planner. Material planner releases the materials of the order. The workflow diagram of order cancellation can be examined in figure 6.23.

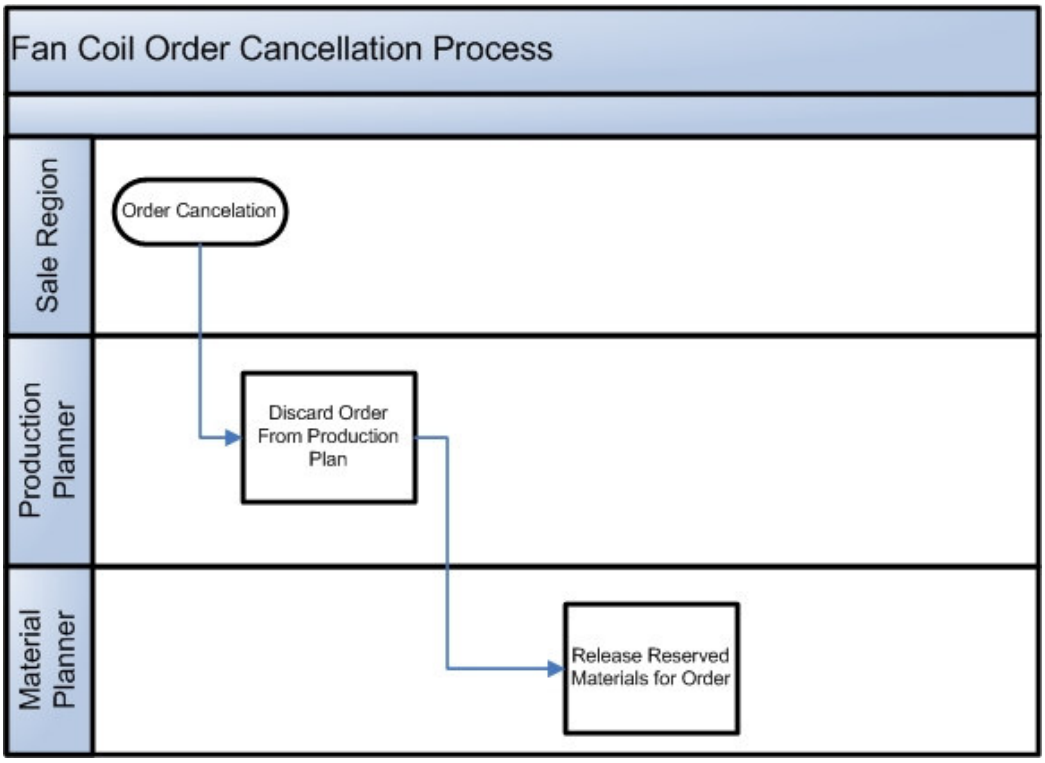

Figure 6.23 Fan coil order cancellation process.

6.3 Fan Coil Product Number Query Process

This process also is simple. A fan coil can have in plenty of possible configurations. Every configuration has a diverse product number. Sale regions sometimes need product numbers they don't have in their database. They send a configuration to designers of the fan coil. The designers return the product numbers. The configuration may not be so definitive that one only one product number is returned. Sale region may not define all the configuration properties. In this case, the possible production numbers are returned with the short definitions of the products. Thus, sale region may enter an order declaration to production planner. The product number query workflow may be examined in figure 6.24.

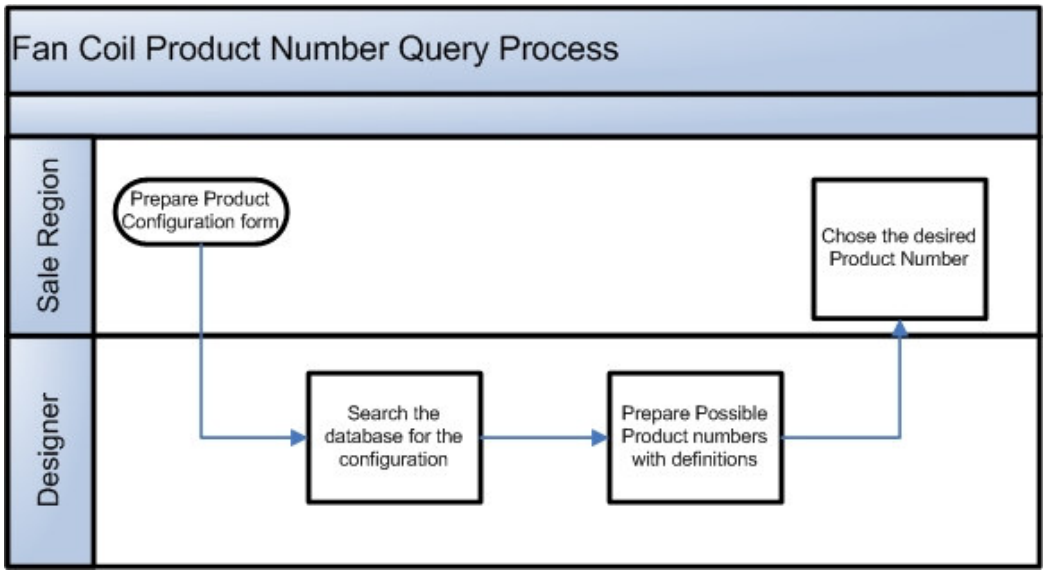

Figure 6.24 Fan coil product number query process.

6.4 Reordering Materials Process

Fan coil department is using a just-in-time system to reduce their inventory levels and associated carrying costs. When using a just-in-time system, you base your purchasing and stock levels on upcoming work, rather than on past usage. There are safety stock levels for materials of fan coil product. Safety stock is the minimum number of the item that you must have on hand at all times. When these levels are reached the reordering procedure must start.

The application built for the system checks the stock levels everyday whether if the stocks are at safety stock limits. If any stock level of a material is at limit, a message is sent to inbox of gui agent of material planner. The other procedure steps are taken care of the user of the material planner.

CHAPTER SEVEN

CONCLUSION

With the growth of the Internet and the World Wide Web, the computation power of software systems is likely to be evaluated on the basis of the power of interaction capability of the systems. Computing is something that happens by communication between computational components. These components are viewed as providing services to one another. They may not all have been designed together or even by the same software development team. In addition to this, these components are not necessarily activated by human users but may also carry out actions in an automated and coordinated manner when certain conditions hold. How should we take this approach into action? The answer lies with agent technologies.

Agents offer a new and more appropriate route to the development of complex computational systems, especially in open and dynamic environments. However, particular tools and techniques are needed. Some of them listed below.

- Methodologies for analysis and design issues are needed. Agent paradigm is different than object paradigm. The methodologies used for object oriented program can not fit to agent programming exactly.
- Agent architectures are needed for the design of individual software components
- Tools and abstractions are required to enable developers to deal with the complexity of implemented system
- Supporting infrastructure must be integrated. The infrastructures for accommodation of agent systems and legacy systems are needed.

Why are agent technologies still used rarely in business environments? There are a number of reasons for this.

- Research in the area of agent technology is also still only in its infancy. If we compare with object-oriented (OO) programming approaches, we will understand why it is so. The initial research on OO programming was in 1962. C++ programming language has come more than 20 years later and first version of Java has come 32 years later. As a consequence of this, knowledge of agent technologies is still not widespread among commercial software developers.
- Because of immaturity of research and development in agent technologies, more proven methodologies, tools are needed. Complementary, products and services like integrating legacy systems with agent systems are needed. When development in agent technologies is satisfactory, costs and risks of agent systems will reduce.
- Many potential applications of agent technologies require the participation of entities from more than one group or organization. Automated purchase decisions along a supply-chain, for example, require the participation of the companies along the chain, so that implementing a successful agent-based application requires agreement and coordination from multiple companies. That's why the current agent systems are occasionally closed systems for one organization.

For these reasons, the agent community is spending so much effort on developing standards for agent communication and interaction, such as FIPA, so that agent systems may interact without the need for prior coordinated technology adoption decisions. More generally, the adoption of agent technologies in business environments depends on how fast and how well agent technologies can be linked to existing and proven software and software methods (legacy systems). Agent technologies should be targeted at those application domains which are proven and tested for a long time. After wrapping the legacy systems, newly built systems may be pure agent systems. Using new technologies without throwing present systems will be the most appropriate choice to firms which don't want to take risk and cost of new systems.

Software programs are being specialized in domains. These specialized domains are getting smaller and smaller. There is no software program that works alone. Every program gets input from its environment and provides output to its environment in some way. The need of interaction and integration is going to be getting more vital. New systems may have good skills on interacting with other systems but legacy systems have always a problem on interacting with other systems. The efforts on wrapping the legacy systems with interacting capabilities will be supported most in next decade. To replace the legacy systems is so risky and expensive. Wrapping with new technologies is the best choice. Agent technology is the best promising technology so far for wrapping problems of legacy systems.

Another key issue of integrating systems is having a tool that solves problems in the way people solve in daily life. If the tools used for integrations force the users to think in another way they are accustomed, the integration process will be inclined to failure. The simulation power of the integration model plays important role for the integrated systems. Multi-agent systems offer strong models for representing complex and dynamic real-world environments. For a reasonable time, companies are using process management methods in their business. The systems that offer process management have great competition opportunity in the market.

The workflow management systems are in widely usage today. Workflow management technologies allow people and organizations to automate, manage and improve the processes that govern interpersonal coordination and collaboration. Many organizations have chosen traditional workflow management systems to automate their processes. However, traditional workflow management systems lack of reactivity, semantics, resource management, heterogeneity, automation and generic interfaces (see section 2.2 for details).

Most of the needs of which traditional workflow management systems lack are met by agent paradigm and standardization efforts on agent communication. Agent based workflow management systems will meet most of the needs mentioned before. Agent based workflow management systems are not only reactive and also proactive.

They are autonomous and social. These capabilities come from their individual components; agents. Naturally they contain the abilities of agents. These capabilities are gathered and implemented in agent development frameworks like JADE. FIPA standards encapsulate standards dealing with agent platforms, interactions between agents and agent platforms. The resource management is standardized in FIPA specifications (by Directory Facilitator (DF) and FIPA communication protocols) Heterogeneity need of the workflow systems is met by FIPA interaction standards between agent platforms.

The semantics and generic interface needs of workflow systems are left. These needs are met when the workflow system is in implementation phase. The semantics need can be met by usage of strong definitive messaging languages like RDF or OWL. Knowledge representation capabilities of messaging language will help organizations to build semantics. The other need of generic interfaces is so important for integrating with legacy systems. The work of integrating agent systems with legacy systems must not be so complex. Otherwise, the cost of implementation will be expensive. Risk of failure will be much more.

The framework, SADE, developed in this study helps its users to build their semantics and integration of their legacy systems. System administrators in firms don't have so much time to deal with design issues about multi-agent systems. Especially efforts on building generic interfaces will increase the usage of agents in workflow and regular systems.

As a result, the companies in every sector choose software solutions having process oriented solutions because they run their business on the process management methods. However, companies do not tend to change their existing systems because of the risk and cost of building systems from scratch. Process management software models which do not push the implementers to leave the existing systems have a good position in market. The best promising solution model is using agent based process management systems so far because agent based

systems have a great simulation power of real complex systems and they have great ability of being wrapper around existing systems.

Following research topics are identified as future research opportunities:

- Modeling: Business processes need to be well described. We need modeling methods especially for the processes among organizations.
- Communication: Efficient communication languages, ontologies for exchanging service definitions.
- Personal working environment: Good user interfaces by using interface agents and personal assistant techniques.
- Integrated with other technologies: Integration with other technologies such as project management, intelligent scheduling, and optimization, etc.
- Learning: Creation of new business process logic during run time through learning.

REFERENCES

Ahmed, K., Ayers, D. & Birbeck, M. (2001). *Professional XML Meta Data*.UK:Wrox Press.

Bellifemine, F., Caire, G., Poggi, A., & Rimassa, G. (2003). Retrieved March 12, 2006 from http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf

Bullock, V., & Cliff, D. (2004). *Complexity, & Emergent Behaviour in ICT Systems*. Retrieved March 14, from http://www.foresight.gov.uk/Intelligent_Infrastructure_Systems/Emergent_Behaviour.pdf

Budimac, Z., Ivanovic, M., & Popovic, A. (1999). *Workflow Management System Using Mobile Agents*, In Lecture Notes in Computer Science, 168-198, Springer.

Chung, P.W.H., Cheung, L., Stader, J., Jarvis, P., Moore, J., & Macintosh A. (2003). *Knowledge-based process management—an approach to handling adaptive workflow, Knowledge-Based Systems 16 (3)* 49–160.

Durfee, E. H. , & Lesser, V. (1989). *Negotiating Task Decomposition and Allocation Using Partial Global Planning*. In Distributed Artificial Intelligence, Volume 2 229-244. San Francisco, Calif.

FIPA (The Foundation for Intelligent Physical Agents) (2002). *FIPA Abstract Architecture Specification*. Retrieved March 23, 2006, from http://www.fipa.org/specs/fipa00001/SC00001L.html

FIPA (2004a) , *FIPA Agent Management Specification* Retrieved March 23, 2006, from http://www.fipa.org/specs/managementspecs.tar.gz

FIPA (2004b), *FIPA Request Interaction Protocol Specification.* Retrieved March 25, 2006, from http://www.fipa.org/specs/fipa00026/SC00026H.html

Garrido, L., & Sycara, K (1996). *Multiagent Meeting Scheduling: Preliminary Experimental Results*. In Proceedings of the Second International Conference on Multiagent Systems, 95-102. Menlo Park, Calif.: AAAI Press.

Genesereth, M. R, & Ketchpel. S. P. (1994). *Software Agents. Communications of the ACM* 37(7): 48-53.

Hubns, M., & Singh, M. (1998). *Workflow agents.* IEEE Computing July– August 94–96.

JADE (2006), Java Aggent Development Framework Group, Retreived March 12, 2006 from http://jade.tilab.com/

Jennings, N, Faratin, P, Johnson, MJ, O'Brien, PD and Wiegand, ME, (1996). *Using intelligent agents to manage business processes*, Proceedings of the 1st International Conference on the Practical Applications of Intelligent Agents and Multi-Agent Technology London, UK, 345-360.

Lei, Y., & Singh, M.P. (1997). *A Comparison of Workflow Metamodels*, Workshop on Behavioral Models and Design Transformations: Issues and Opportunities in Conceptual Modeling at ER'97, Los Angeles: CA.

Lewis. C. M., & Sycara. K (1993). *Reaching Informed Agreement in Multispecialist Cooperation.* Group Decision and Negotiation 2(3): 279-300.

Luck, M., McBurney, P., Shehory, O., & Willmott, S. (2004). *Agent Technology: Overview and Consultation Report.* Retrieved March 15, 2006 from http://www.agentlink.org/roadmap/al3rm.pdf

Manola, F. & Miller, E. (2002), *RDF Primer*, W3C Working Draft Retrieved March 23, 2006, from http://www.fipa.org/specs/managementspecs.tar.gz.

O'Brien, P.D. & Wiegand, W.E. (1998a). *Agent based process management: applying intelligent agents to workflow, The Knowledge Engineering Review 13 (2)* 1–14.

O'Brien, P., D., & Wiegand, M., E., (1998b). *The Knowledge Engineering Review* , Cambridge University Press, Vol. 13.2, 161-174

Shen, W., Norrie, D.H. & Barthes, J.P. (2000). *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing*, Taylor and Francis, London: UK.

Sheth, A, (1995). *Workflow Automation: Applications, Technology and Research Tutorial Notes from SIGMOD Conference, May*.

Sycara, K. P. (1998). *AI magazine Volume 19, No.2* Intelligent Agents Summer 1998.

Takeda, K., Inaba, M., Sugiara, K. (1996). *User interface and agent prototyping for flexible working*, IEEE Multimedia 3, 40–50.

Turoff, M., Hiltz, S.R., Bieber, M., Fjermestad, J., Rana, A. (1999), *Collaborative discourse structures in computer mediated group communications*, Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences 5–8 January 9.

Vollmann, T., Berry, W. and Whybark, D. (1992). *Manufacturing Planning and Control Systems*. Irwin: New York.

Vaucher, J. & Ncho A. (2003). *JADE Tutorial and Primer*. Retrieved March 6, 2006 from http://www.iro.umontreal.ca/%7Evaucher/Agents/Jade/JadePrimer.html

Workflow Management Coalition, (1996). *Workflow Management Coalition Terminology and Glossary, Doc. No. WFMC-TC-1011*, Issue 2.0, June 1996.

Wooldridge M. (2000).*Reasoning about Rational Agents*, Cambridge:The MIT Press.

Yan,Y., Maamar, Z. & Shen, W., (2001). *Integration of Workflow and Agent Technology for Business Process Management*, The Sixth International Conference on CSCW in Design.

Zambonelli, F. & Parunak, H. V. (2002). *Signs of a revolution in computer science and software engineering, in P. Petta, R. Tolksdorf and F. Zambonelli (Eds.), Engineering Societies for the Agents World, Lecture Notes in Artifi cial Intelligence* 2577, 13–28, Springer.