SOFTWARE AGENTS FOR DLNET CONTENT REVIEW: STUDY AND EXPERIMENTATION

SEEMA MITRA

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Science and Applications

Dr. Saifur Rahman, Co-Chairman
Dr. Csaba Egyhazy, Co-Chairman
Dr. William Frakes, Committee Member

September 8, 2006
Falls Church, Virginia

Keywords: Software Agents, Digital Libraries, Content Review, Performance Comparison

SOFTWARE AGENTS FOR DLNET CONTENT REVIEW: STUDY AND EXPERIMENTATION

SEEMA MITRA

ABSTRACT

This research is an effort to test our hypothesis that a software agent based architecture will provide a better response time and will be more maintainable and reusable than the present J2EE based architecture of DLNET (Digital Library Network for Engineering and Technology). We have taken a portion of the complete DLNET application for our study, namely the Content Review Process, as our test bed. In this work, we have explored the use of software agents in the current setup of DLNET for the first time, specifically for the Content Review part of the application and tried to evaluate the performance of the resulting application. Our work is a novel approach of doing content review using software agent architecture. The proposed system is an automated process that will asynchronously look for suitable reviewers based on content (the input) and create logs for the administrator to view and analyze. In the first part of the thesis we develop a new system that is parallel to the existing DLNET Content Review Process. In the second part, we compare the newly developed Content Review Process with the baseline (old Content review Process) by designing comparison tests and measuring instruments. This part of the thesis includes the selection of dependent variables, design of various measurement instruments, execution of the quasi-experiments and analysis of the empirical results of comparisons tests. The quasi-experiments are done to measure the response time, maintainability, scalability, correctness, reliability and reusability of the two systems. The results show that the proposed software agent based system gives better response time (an improvement ranging from 57% to 82%) and is more maintainable (an improvement ranging from 16% to 67%) and more reusable (an improvement ranging from 1% to 26%). The improvement in the response time may be attributed to the fact that the agent based systems are inherently multithreaded while the existing content review system is a serial application. Both the systems, however, give comparable results for other dependent variables.

ACKNOWLEDGEMENTS

I wish to express my gratitude to Dr. Csaba Egyhazy, my research advisor, who has been most supportive and has guided me throughout my study giving me direction whenever I got lost. I would also like to thank Dr. Saifur Rahman for giving me permission to use all the resources I needed at ARI (Advanced Research Institute, Arlington, Virginia) and for all his advice, guidance and considerations. I also want to thank Dr. William Frakes who has been very helpful, providing all the support and resources and very valuable suggestions and feedback on my work. His guidance paved the way for the second part of my work in designing the performance comparison plan. I want to thank you all for your patience and support.

I would also like to thank Mr. Yonael Teklu for his prompt help and support whenever I asked for it and would also like to mention and thank Latricia Nell for her immediate response whenever I needed her support.

I am very grateful to my friends Gaurav Aggarwal, Vaishali Sharma, Sarala Sriram and Pankaj Aggarwal for helping me out with the project. They helped me with the maintainability and reusability tests for the two applications.

Finally I would like to thank my husband, Sandeep, my sons, Siddharth and Sarthak, my sister, Reena and my Mom for letting me work on my thesis and being so patient with me, for being such a great help and being so encouraging always.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

1. Introduction

In this chapter, we describe the overview and motivation for this thesis. We also highlight the approach of the study and the contributions we have made.

1.1 Overview

Most of today's applications are driven by their successful integration with the internet. With the advent of World Wide Web, the digital libraries have become very popular. Digital Library Network for Engineering and Technology (DLNET) has been developed as part of National Science Digital Library (NSDL) initiative of National Science Foundation. The digital library is in place and is being used by users worldwide, and now we want to direct our efforts towards improving it in terms of ease of use, applicability to a diverse base of users, and optimizing the performance. Digital libraries are a growing concept especially in today's academic world. Research efforts are abound to better the services, integrate the different digital libraries and optimize their performance. Nabil Adam and Yelena Yesha in [26] provide a complete summary of the conceptual comparison of digital libraries with electronic commerce and highlight the various research issues in the areas. Digital libraries are geared towards networking of digital resources and thus making them available for the users irrespective of the location of the users or the resource. Once the digital libraries become popular and resources become available, the next phase of challenges is to make them more efficient. Robert L. Grossman [27] talks about the data retrieval challenges in digital libraries. For now, most of the digital library use is limited to getting information from a single resource at a time. Next, we'll want to retrieve information from a cluster of related resources as opposed to single resource search that we do now. This work is a step towards exploring the use of software agents in making DLNET more usable and efficient. A lot of research is being done to optimize the digital libraries; e.g., Martin in [38] proposes a three layer architecture for digital libraries and highlights the importance of virtual libraries.

From the user's perspective, the most important services a digital library can offer are the following:-

- A good (relevant) collection of resources.
- A convenient and effective way to search for the relevant resource.

In addition, the other services for effective usage of a digital library are:-

- Capability to store a variety of resources.

- Connecting with other digital libraries to offer the users with a vast collection of resources which is not possible for one digital library to cater to.
- Since the application of a digital library will keep growing, it will most likely be hosted as a distributed application in the future.

Keeping these features of digital libraries in sight, we are experimenting with the introduction of software agents in DLNET. Sanchez, Leggett and Schnase [28] have proposed architecture for digital libraries based on software agents and the work has been a successful integration of the two concepts (digital libraries and software agents). Driven by the literary efforts mentioned above, our motivation for using software agents for DLNET is driven by the following possible gains.

- Software Agents can provide advanced document processing capabilities to analyze the text in the documents.
- The growing amount of information will require advanced infrastructure and means for organizing and accessing information. This can be done through dedicated information agents.
- Information seeking agents can be employed to seek information from the World Wide Web.
- Asynchronous communication possible with software agents can handle the events like submission of resources and reviews without intervention of the administrator.
- Software agents can offer personalized services to a user (of the digital library) by maintaining a user profile and making intelligent decisions based on the same.
- Software agents can be used to optimize the selection of the reviewers for a resource submitted based on reviewer data and relevance to the resource submitted.

We believe that the use of software agents in DLNET provides the right foundation on which to build all the services we mentioned and so many others that are possible too.
In this thesis, we propose to setup an architecture based on software agents that is adaptable, extensible, efficient and flexible.
Regular libraries lead to digital libraries, and the agent based digital libraries seem like a natural extension. In using agent technology, our goal is to make DLNET a digital library system that provides value added services to the users and provides a uniform interface for heterogeneous information sources. This project is the first step towards that goal.

This investigation has two major goals –

1. First, to design and implement a software agent based content review process for DLNET - DLNET already has a process for the selection of reviewers based on the content submitted and the reviewers available. The current process involves manual intervention of the administrator to start the selection. We are developing a system consisting of software agents that will do the same automatically. Since we want to study the performance of both the systems, we want to run them in parallel and compare their performance test results.

2. Having developed the software agent based system, design quasi-experiments to compare the performance of the two systems, namely the existing java based with human-in-the-loop system and the software agent based system, and draw conclusions. This part of the work compares the performance of the two approaches to assess whether the proposed architecture would handle the expected workload better. The traditional approach for this type of comparison is to develop the new software system and collect performance measures on both the existing system and the new system in development environment as given by Ann Blandford, Suzette Keith, Iain Connell and Helen Edwards in [25] where they give a comparison of four different techniques for comparing the usability of digital libraries.

1.2 Motivation and Problem Statement

DLNET is envisaged as a platform for information discovery, interaction, content building and distribution that will support pedagogy and learning in Engineering and Technology [29]. Now, we are looking for ways to better the current setup and optimize the various processes being followed in DLNET.

For this thesis, we are particularly focusing on the process of content submission. The process of the DLNET resource submission is like this – There is user of type Contributor who submits the resources and the related metadata for storage in DLNET. Before these resources can become part of the DLNET repository, they are peer reviewed. The reviewers for doing the peer review are selected from the DLNET database. DLNET has users of different interests and backgrounds associated with it. The selection of reviewers is dependent on the interests of the reviewer, the background of the reviewer and the past relation with DLNET.

In this work we are trying to implement the selection of reviewers through the use of software agents in an effort to experiment the use of this technology to confirm their applicability and efficiency in such a scenario. Software agents are capable of autonomous and asynchronous function and seem to be quite appropriate for the problem at hand. This work is an effort to establish the same by first implementing an agent system, and then comparing it to the current system.

1.3 Approach

This work is composed of two serial activities. First is the study and implementation of the software agent system for Content Review of DLNET. The second part consists of experimentation design and analysis to compare the performances of the agent-based and the current systems. We first study the various agent architectures and select the open source *JADE* architecture. We then implement the Content Review activity of DLNET in parallel to the current application and collect a log of the attributes like response time, failures and results of the process. We also log the response time and other performance attributes of the current application to have a comparison of the two.

This is how the rest of the report is organized – Chapter 2 is about the literature research and review that has gone into this study and how we select one of the architectures. It also mentions the relevant literature. Chapter 3 focuses on the selection of the agent architecture, and briefly describes the various architectures. Chapter 4 focuses on the DLNET Content Review Process, what is the proposed architecture and how it adds to the current system. Chapter 5 describes in detail the proposed architecture based on agents, the next chapter details the test plan for performance comparison. Chapter 7 presents the results of the different tests done. This report concludes with chapter 8 highlighting the conclusions derived from the work, and the possible future research directions.

1.4 Thesis Contribution

A novel approach to content review for digital libraries
As part of this thesis, we study a few digital libraries (e.g., University of Michigan Digital Library, Zuno Digital Library, and Next Generation Digital Library) associated with academic institutions that are implemented using various technologies including the digital libraries using software agents. In addition, we survey the Information Retrieval/Digital Library literature, but do not come across any digital library employing software agents for the selection of reviewers based on the content submitted. Our work therefore is a novel approach of doing content review using software agent architecture. The proposed system is an automated process that will asynchronously look for suitable reviewers based on content and create logs for the administrator to view and analyze.

Design and experimentation for performance comparison
DLNET is a complete application that manages users, resources and provides services to the users. Selection of reviewers when content (a resource) is submitted is part of the complete application. We implement a software agent based application to automate and optimize the selection of reviewers in DLNET. To justify the proposed architecture, we conduct extensive performance tests to compare the existing content review application with the one reported herein. We adopt an experimental approach for the performance comparison of an agent based application with the existing non agent based application. The guidance for the performance comparison plan comes from [24]. We design quasi-experiments, create test data and add features in both the applications to collect data and then compare their performance based on the quality attributes selected and the test plan designed based on requirements.

A basis for future work
This thesis lays a foundation for the future expansion of DLNET using software agent architecture. Future research studies at Virginia Tech can augment and extend the work done here as it establishes a ground for adding new services for the digital library.

CHAPTER 2

2. Background

This chapter presents the review of the literature. It briefly describes the different digital libraries we study and the different agent architectures we consider for our work.

2.1 Overview

Digital libraries are a relatively new concept for information sharing. The research efforts are directed towards networking of digital libraries, making them usable for different formats of resources, and towards increasing the efficiency and usability. Almost all universities and educational institutes imparting advanced education are geared towards sharing the academic resources on the World Wide Web. There is a simultaneous effort to link these different libraries so as to avoid duplicate resources being stored and also to increase availability. As part of the research initiatives to make the digital libraries more efficient and useful for the users, our work is directed towards the study of digital libraries and software agents. We begin by studying different software agent architectures to select the one that best suits our requirements.

Section 2.7 gives an overview of the design of experimental setup used to compare the two applications and their architectures, i.e. the existing content review application running on a client-server based architecture, and the proposed software agent based application running on an agent-based architecture.

2.2 UMDL (University of Michigan Digital Library) Agent Model

We study the software agent based architecture of UMDL (University of Michigan Digital Library), which is a very effective, efficient, extensible, distributed system. Managing a digital library poses a lot of problems. It's a big repository and efficient management of resources, users and services is required. One option is to have manual administration like in regular libraries. On the other hand, this administration can be incorporated in the software. With this in mind, UMDL uses software agents that buy and sell services from each other. This model is based on market based multi agent system [4], [5].

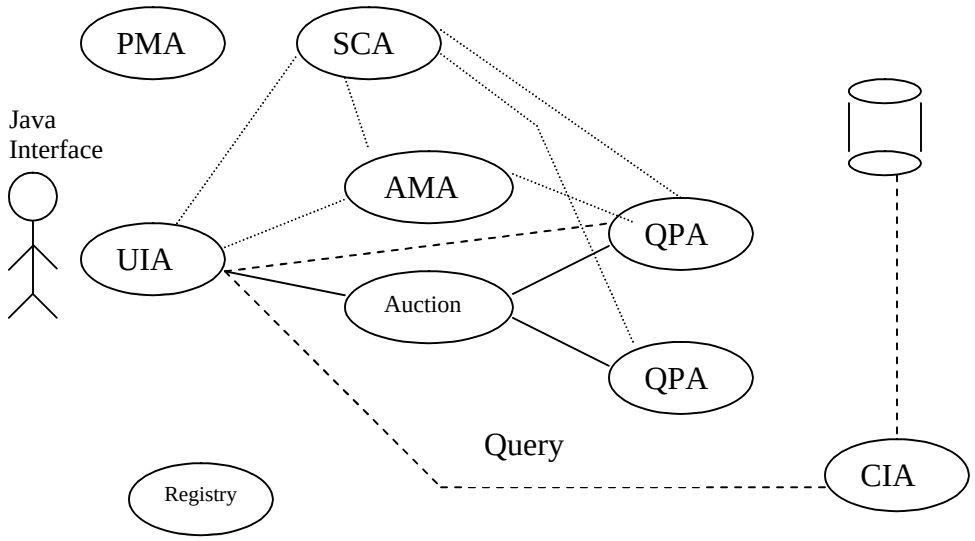

Figure 1: The UMDL Service Market Society [4]

Figure 1 illustrates the UMDL architecture. There are different kinds of agents –

- User Interface Agents (UIA) – The user preferences and desired services and managed by these agents.
- Query Planning Agents (QPA) - These agents act as middlemen between user queries and the results form the library.
- Service Classifier Agents (SCA) – Used for registration of agents in a central directory.
- Collection Interface Agents (CIA) – Provide access and search facility for the collection of resources.
- Auction Manager Agent (AMA) – These agents help the buyers and sellers of services find each other.
- Price Monitoring Agent (PMA) – These monitor the price of query planning services.

The agents first register themselves with the registry, then the agents wanting to buy the services get a list of such sellers through AMAs, the buyer then contacts the seller which in turn uses CIAs and QPAs to get the results.
UMDL is a layered architecture that is very flexible, scalable and extensible.

2.3 ZunoDL (Zuno Digital Library) Agent Model

Next, we study the architecture of another agent based digital library, ZunoDL (Zuno Digital Library) in [2] and [6]. This is a collection of tools and techniques for building digital libraries as agent based information economies. The various components of this system are owned by different groups and they collectively form a distributed system. ZunoDL is driven by making commercial digital libraries that can overcome the problems of search engines by providing a framework for varied real world

applications related to digital libraries like distance learning, electronic commerce etc. It is also software agent based framework that lets users create custom information economies by putting together a set of agents. This framework caters to the following three types of users:-

- Consumers – Represented by User Interface Agents (UIA), the users who want to access resources.
- Producers - Represented by Library Service Agents (LSA) and Catalogue Agents (CA), these users/authors who want to share their content.
- Facilitators – Represented by Search Agents (SA), it is the ZunoDL service network that maps consumers and producers with each other.

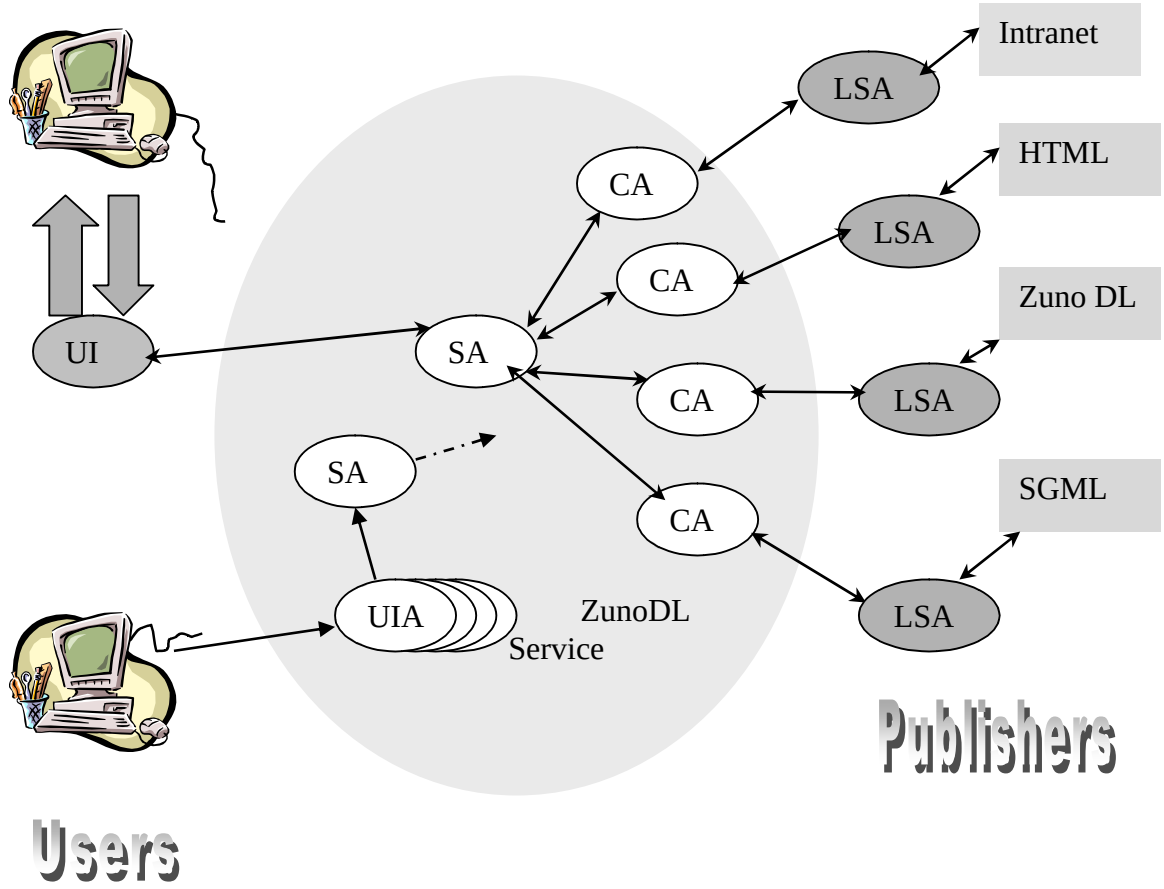

Figure 2: ZunoDL System Framework [6]

This is a much more spread out and distributed application although the basic agent architecture is the same as that of UMDL. This can be used to implement individual digital libraries over any kind of network, internet or intranet. The basic difference when compared to UMDL is that UMDL is a centralized standardized architecture and all users (publishers or consumers) become part of one big library whereas ZunoDL provides an opportunity for a consumer and publisher pair to create their own private digital library over a network.

2.4 Next Generation Digital Library Project

This is another project of developing a digital library based on software agents that we study for our work on this thesis in [10] and [12]. This project is based on a distributed object oriented 3-layered architecture using CORBA and agents. The basic architecture consists further of three sub-architectures –

- Messaging Architecture
- Agent Architecture
- Database Architecture

The agent architecture provides interoperability, rapid application development and automation. We also adopt the agent architecture for DLNET for these reasons. The basic agent infrastructure provides services like Lifecycle Management, Agent Communication, Security, Directory and Migration.
With the increase in the volume of resources and more efficiency required in retrieving the right content, this architecture depends on the software agents to provide the extensibility and interoperability to communicate in heterogeneous environments. The basic infrastructure of the next generation digital library is provided by CORBA and J2EE and software agents are used to implement the advanced requirements.

2.5 Cougaar Agent Model

Cougaar (Cognitive Agent Architecture) is a very efficient and effective platform for distributed agent based systems. It is a Java Agent, hierarchical architecture designed to support data intensive, inherently distributed and highly scalable applications [16].
Intra-agent communication happens with the use of a local blackboard that helps reduce latency. Applications in Cougaar are designed around agent communities. Salient features of the architecture are:-

- The computation in a Cougaar agent is done by Plugins, which are software components that define the behavior of the agents.

- A Blackboard is an agent-local memory store that supports publish/subscribe semantics. The components or Plugins can add/remove/subscribe objects from the blackboard.
- Inter Agent communication in Cougaar takes place with the allocation of tasks.
- Intra Agent communication takes place with the subscriptions/ publications to the local blackboard.

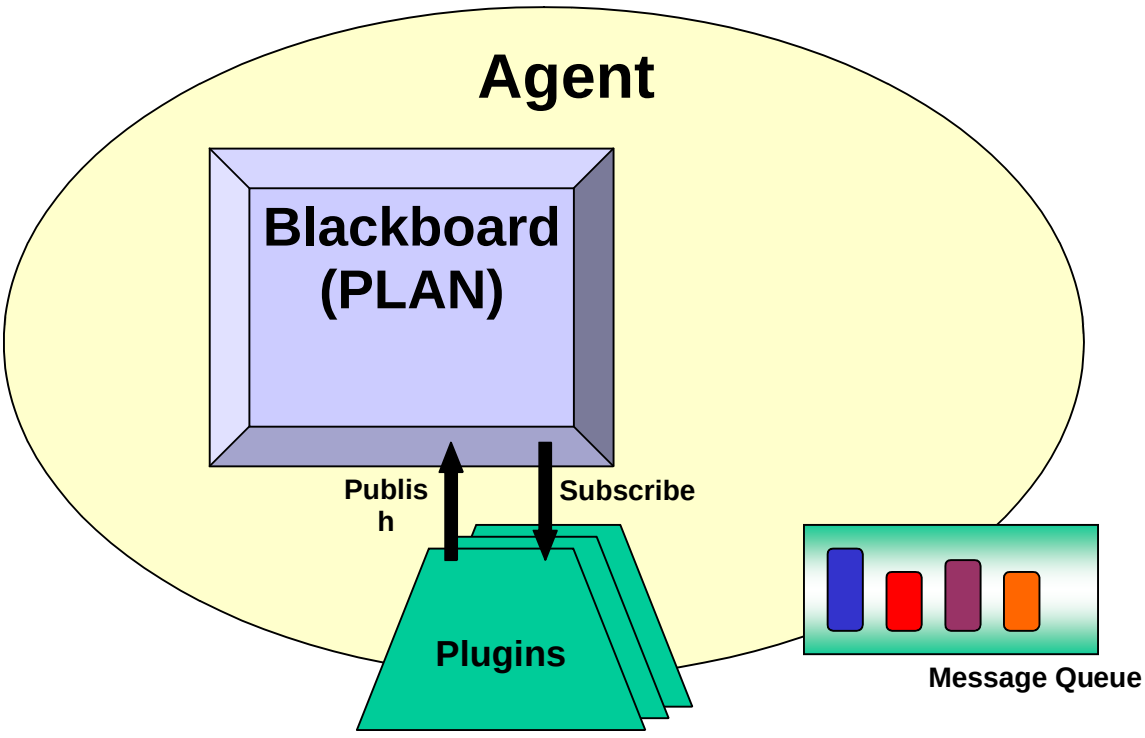

Figure 3: Cougaar Architecture [32]

Cougaar is most suited for hierarchical planning problems and does not offer any advantage for a problem like ours. Cougaar is not an easy architecture to follow, and writing applications is far from trivial. Since the project reported herein is the first step in the use of agent technology for DLNET at Virginia Tech, and we anticipate many other research students to contribute to it in the future, we want to select a platform that is well documented and easy to use.

2.6 JADE Agent Model

We then considered and studied the open system FIPA (Foundation for Intelligent Physical Agents - http://fipa.org/) compliant, java based JADE (Java Agent DEvelopment Framework) architecture in [14] and [15]. This architecture is very well documented, is efficient and it takes very little time for a user to get it up and running.

JADE based applications have a main container (the runtime environment) and an infrastructure with agents for communication (ACC), directory service (DF), and naming and management services (AMS – Agent Management System)

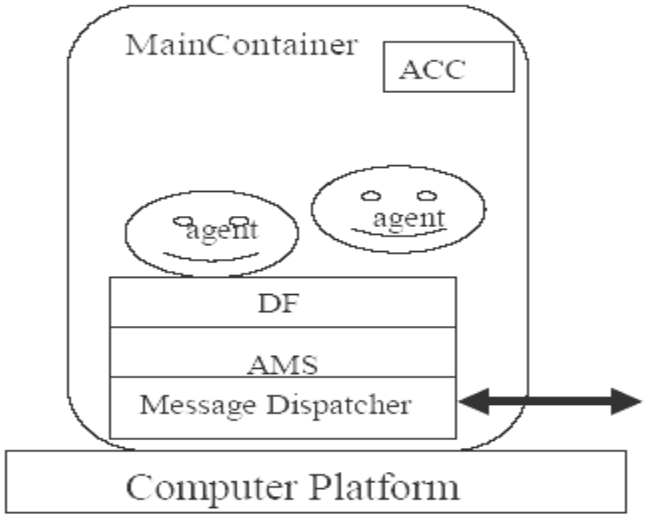

Figure 4: The Jade Main Container [15]

Figure 4 shows a jade container with the infrastructure provided agents and the application specific agents. The following agents are provided by the JADE infrastructure and are used by our application.

1. *The Agent Management System* is an agent that is part of the main container and it provides the naming service to the agents that get added in the application. It is required that every agent has a unique name in a platform. To register a new agent, we define the type of the agent and give a unique name to it.
2. *The Directory Facilitator* provides a service to help locate other agents. For this service the agents need to have a type associated with them. When searching for a type of agent, we define the type of agent we want to locate and the Directory

Facilitator gives the name of the agent if some suitable agent is registered in the platform.

2.7 Performance Comparison Methodology

Our efforts are guided by the fact that the software architectures of today are moving towards component based systems and in that light, our effort to specifically measure the performance parameters of a component based architecture and a non component based architecture are useful. We review some literature including [7], [8], [13], [31] on how to go about doing a performance measurements of software architectures. Although we find that there is relatively very little published work comparing the performance of software agents vs. traditional software systems, we study the following literature to help develop our comparison plan:-

- Michelle Casagni and Margaret Lyell in [7] compare two components based architectures and do a good comparison from the high level basic framework, the application specific comparison and the design attributes. As in our work, this study compares non-agent architecture with agent based architecture. This is probably the only work we see that develops a comparison structure and does a detailed comparison. But it differs from what we do in that we compare a FIPA compliant architecture with a non component based one.

- Filippos I.Vokolos and Elaine J. Weyuker in [8] give guidelines about how to go about doing a performance testing of a software system. Although it deals with the telecommunications industry, it nonetheless provides direction about how to plan the performance testing for our case, and how to create test data; keeping in mind that it should be representative of the actual load to be experienced in the live scenario. Guided by this paper we create test data for average workload conditions and heavy workload conditions. This reading covers a lot of performance parameters and guides us in considering the different aspects of performance of a system like resource usage, throughput, queue lengths etc. We have selected the ones relevant to our case.

- This reading by Giovanni Denaro, Andrea Polini, Wolfgang Emmerich in [13] gives an insight into planning the performance testing of applications at software planning and architecture definition stage. The reading is applicable for distributed applications that use some middleware that is available off the shelf and that can be tested before it is integrated with the application in question. We could have used the methodology given in this reading if we were planning both the applications (the existing baseline application and the proposed agent based application) from scratch. Here we already have a live application and so the conditions do not apply. We design a test plan and measurement quasi-experiments of our own that suit our requirements better.

Efforts mentioned in the literature above have either involved performance measurement of a single system or, as in the case of [7], the architectures have been both component based. Guided by the metrics measurement covered in [24], we develop a test plan suited to our situation. Our comparison plan, although influenced by the readings above, is different because it compares a component based FIPA compliant framework with a serial java application. More details of our performance measurement plan and methodology are found in section 6.1.

CHAPTER 3

3. Selecting an Agent Architecture

3.1 Introduction

In this thesis we have integrated software agents with DLNET (The Digital Library of Virginia Tech). For this, we first select the characteristics and features we should concentrate on for our experiments for the content review process of DLNET. Then we study agent architectures such as JADE and Cougaar. We also look at the agent architectures of current digital libraries using agent based services and features such as the UMDL (University of Michigan digital Library), ZuNO Digital Library.
We study all these available architectures and compare them with respect to our requirements to finally decide on using the agent infrastructure of JADE as the chosen middleware for our study.

3.2 Desired features of the selected architecture

We start with listing the features we want the new architecture to have

1. Interoperability – Ease of connection with other open archive digital libraries.
2. Extensibility – Ability for step by step extension to the system.
3. Ease of deployment.
4. Available documentation/ support on understanding and setup of the architecture.
5. Relevance to our problem domain.
6. Easy Maintenance

We study the existing application and how content review is handled, following it up with discussion with DLNET team members to arrive at important features we want to work on.

3.3 Selecting an Agent Model

Software agent architecture is based on peer-to-peer computing as opposed to the popular client server paradigm. The advantage of using peer-to-peer communication is that all the modes of the application can communicate with each other. The facility of looking for an appropriate peer to communicate with is separately managed. In client server model, the client has to know the exact details of the server it wants to communicate with and the communication details have to be finalized in design time

itself. Software agent model is flexible in the sense that once an agent specifies a requirement, any other agent fulfilling the requirement can respond and the two agents can communicate. These service provider agents can even be created at run time. There are applications where client server models will not suffice and so agent based solutions are more appropriate. These reasons and benefits of peer to peer technology based software agents made us select our architecture.

We selected JADE as the agent architecture to build the relevant agents for the content review process of DLNET because,

- it had all the agent features that we needed (and more)
- communication between student "agents" running on various workstations on the network was trivial to do
- it was efficient and robust enough to tolerate some common programming errors
- it followed FIPA standards
- the user group is very active and implementers typically respond to problems within 24 hours

In selecting the JADE agent architecture for our application, we compare it to Cougaar, UMDL (University of Michigan Digital Library) architecture, Zuno digital library and Next Generation digital library architectures.

One of the important characteristics we look into the architecture is ease to understand and deploy. JADE and Cougaar are equally easy to deploy but to understand JADE and get started with it is much faster than Cougaar.
UMDL and ZunoDL were quite easily expandable but the initial deployment was relatively heavy and needed a lot more resources than JADE.
UMDL and ZunoDL are basically developed to be used by multiple computers across geographical locations. The storage of resources also happens at multiple locations, whereas our system is already in place and is housed at a single place. The conceptual model is different as we are catering to users spread out in geographical location but the storage and the application code is hosted at one place. There is, therefore a different requirement for DLNET and the architecture of these two digital libraries is inappropriate for our cause.
In the model being used by DLNET, the library is to be linked with other digital libraries under NSDL. To enable this was an important requirement for our project. The architectures of UMDL and ZunoDL are self contained and although they are expandable, they can’t be linked easily with other digital libraries as they follow a different conceptual model. So we look for other alternatives.
JADE as a software agent platform is very popular and is used in many research studies in many well known European universities. The users of JADE maintain a very lively mailing list and respond to queries very fast. So this architecture is very easy to understand and we find it to be very maintainable.

In addition to providing a runtime environment and a library of classes, JADE also provides graphical tools for monitoring the activity of all the registered agents.

Following table summarizes the comparison of various architectures

FEATURE	UMDL	ZUNODL	NEXT GENERATION DIGITAL LIBRARY	COUGAAR	JADE
Interoperability	Good	Good	Good	Good	Good
Extensibility	Good	Good	Good	Good	Good
Ease of deployment	Not Easy	Not easy	Not easy	Good	Good
Available documentation	Fair	Fair	Fair	Not good	Good
Relevance to our problem domain	Not much	Not much	Not much	Good	Good
Easy maintenance	Not Applicable	Not Applicable	Not Applicable	Fair	Good

Table 1: Summary of the comparison of various architectures we considered

CHAPTER 4

4. Modifying the DLNET Content Review Process

4.1 Introduction

DLNET is a stable application and is being used by users worldwide. The application is a component based J2EE application. In this thesis, we investigate ways to improve the content review portion by suggesting and implementing an alternative architecture that employs software agents.

4.2 Existing Content Review Process

DLNET has three different types of users:-

1. Contributors – Users who submit Learning Resources
2. Reviewers – Users who review the submitted resources before the resource gets added to the repository.
3. Users – Users who use the library for browsing or searching.

The content review process starts with a contributor submitting a learning resource. A learning resource could be a document, image, presentation etc. The contributor needs to add some information about the resource like the category and the type of resource. This metadata for the resource is clubbed with the resource by a DLNET utility. A check is performed to make sure that all the necessary information is given by the user at the time of submission. Also the integrity of the package is checked to make sure all the referenced objects are included in the package. Once all the checks are done, the resource and the metadata (in an XML file) are stored in a temporary repository.

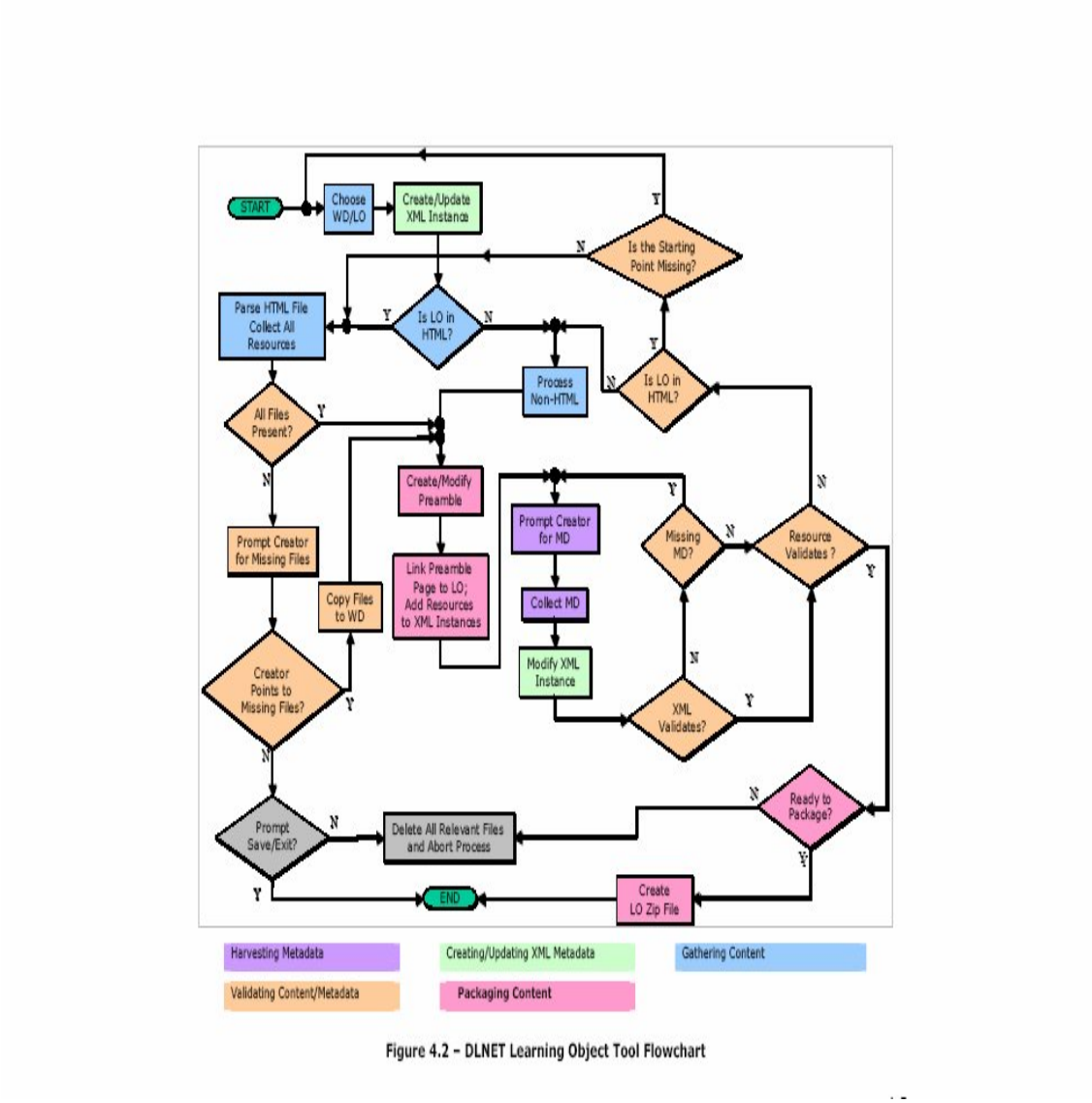

Figure 4.2 – DLNET Learning Object Tool Flowchart

Figure 5: Learning Object Tool Flowchart showing the Content Review Process Flow

[Final Report at http://www.dlnet.vt.edu]

The existing Content Review Process at DLNET is a java application which uses various classes of the DLNET infrastructure. Currently what happens at DLNET is the following - the resources get submitted, the administrator keeps a check and when the temporary repository has some resources that need to be peer reviewed, the administrator starts the utility that finds the suitable reviewers for the content submitted. This is done based on the Meta data submitted along with the Learning Resource.

4.3 Proposed Content Review Process

The proposed Content Review System is a JADE application that is always running that is, the agents are always in an active state. One of the agents is responsible for regularly checking if any new content is submitted to the DLNET repository. When some content is found, the content review process starts by sending of messages between agents. When a suitable reviewer is found, logs are generated and the reviewers can be informed accordingly. The details of the system are given in chapter 5.

4.4 Expected Gains/ Performance enhancements

A more extensible and maintainable application is expected as a result of these quasi-experiments.
We have added the following features to the existing application:-

- The content review process is now fully automated.
- The logs are prepared about daily results. It will help in finding out the areas where DLNET does not have enough reviewers and in future, the application can be extended to find out more reviewers in those areas.
- The application architecture is very easy to understand, it is easy to add new functionality to the system.
- An algorithm for a part of the application is easy to replace with a better alternative.

Digital Libraries rely on content submitted by the users registered as contributors. With the popularity of digital libraries, the content submitted will keep on increasing and so will the frequency of submission. We therefore need to make our system scalable and extensible to handle that kind of load. As the number of resource submissions increase with the popularity of digital libraries, a scalable, extensible system is desirable.

CHAPTER 5

5. The proposed Architecture

5.1 Introduction

High module cohesion and low module coupling is the rationale for most structured design methods. Good internal structure leads to good external quality. This is the basis of the design of the proposed system for DLNET content review process using the JADE architecture and software agents.

The JADE architecture provides basic services for running the software agent platform. It is a java application and we can easily integrate it with our existing application.

Each running instance of JADE runtime environment is called a Container as it can contain several agents i.e. provide support for many agents. The first container to start must be the main container. If the application has any more containers, the additional containers have to register them with the main container. The main container holds two special agents that get started with the main container. They provide essential services for the operation of the application. [Refer Figure 11]

These built in agents that provide Directory services and Agent Management services are:-

AMS – Agent Management System – This agent provides the naming service by ensuring that each agent has a unique name.

DF – Directory Facilitator – This is to locate an agent and the agents are identified by the services they provide. In other words if an agent wants to use the services of an agent of type doctor, the DF will help locate a doctor type of agent.

After studying the current system as it works, we design the application by modularizing the components of the system and design the following agents. These agents run on the JADE platform and use the library of classes that JADE provides to together offer the complete application.

AgentStarter – This is the agent that starts the content review process. This agent starts the execution every day. The frequency of the computing can be set by a parameter in the Agent code. This agent sends a message to the AgentContent to start looking for a content to be reviewed.
We add the configuration parameters as constants in this agent code so that these parameters can be changed as and when required.

AgentContent – This agent after getting a message form AgentStarter accesses the database and looks for appropriate status of content that needs to be peer reviewed. If the agent finds any such content, it sends an appropriate message to AgentReviewer. This agent sends as many messages to AgentReviewer as the number of resources found.

AgentReviewer – This agent gets the message and starts to find the reviewers appropriate for the content based on the rules defined similar to the rules in the existing application. On finding the reviewers, this agent sends message to AgentEmail and updates the database accordingly.

AgentUser – If Agent Reviewer cannot find any reviewers in the normal database of reviewers, this agent gets a message and looks for inactive users in the reserve database of reviewers and gets the appropriate users.

AgentMonitor – We have this agent to log the response time of the process. This agent gets messages with the start and end of the process and logs them in a text file. The difference of those times gives the response time for that run of the application.

AgentEmail – This agent is responsible for sending appropriate messages to the selected reviewers.

AgentErrorLog – This agent gets messages whenever there is some error in the processing and logs the errors in a text file.

The code implementing the AgentStarter is given in Appendix D.

5.2 Process Flow

The following figure shows the interactions between different agents and the database. This flow shows only the application specific agents.

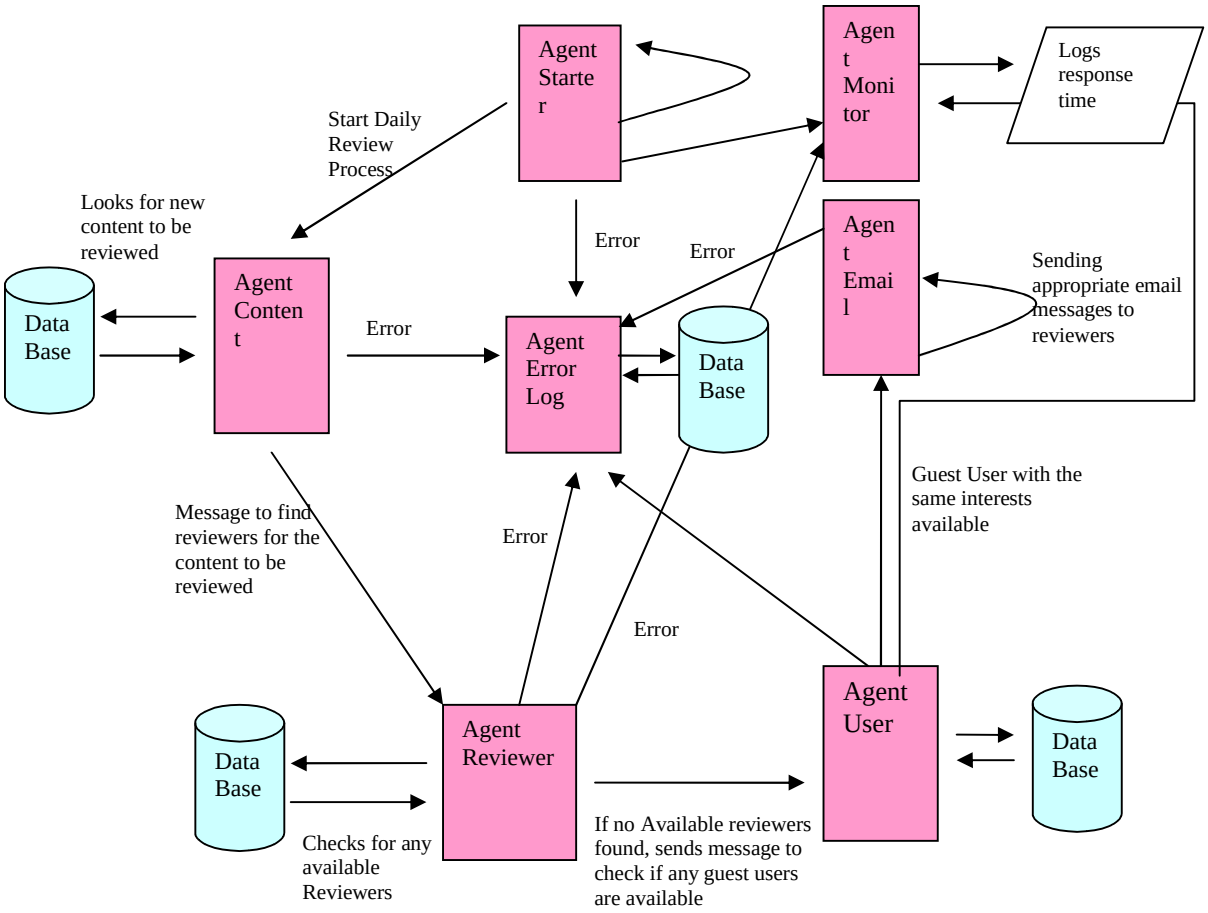

Figure 6: Process Flow of DLNET Content Review in the Agent Based Application

5.3 Running Jade Agents

JADE Application Startup

The software agent application for DLNET content review using JADE architecture is in place and is integrated with the DLNET application. It is a separate application that needs to be started individually.
Once we compile the application including the JADE agents and the code integrating it to the DLNET application, we have to start the JADE platform and run the application specific agents.
To start the JADE platform, the following command needs to execute after setting the classpath appropriately to include all the class files:-

java jade.Boot Starter:AgentStarter Content:AgentContent Reviewer:AgentReviewer User:AgentUser Email:AgentEmail Monitor:AgentMonitor Error:AgentError

This command will start a JADE main container; will initialize the infrastructure agents (the Directory Facilitator and the Agent Management System) and the specific agents we give in the command.

This keeps the container alive till the container is killed specifically. Our application is designed in a way that the Starter agent will keep on initiating the content review process as per the set frequency. The frequency of this review process can be set by changing the constants in the AgentStarter class file. The other parameters can also be changed in the same file, the parameters like location of XML files, the error messages to be reported etc. This is like having a configuration file where all the dynamic information is kept and can easily be managed.

The screenshot below shows the messages being exchanged between various agents of the application. This kind of visual aid helps understand the operation and also helps in debugging the application.

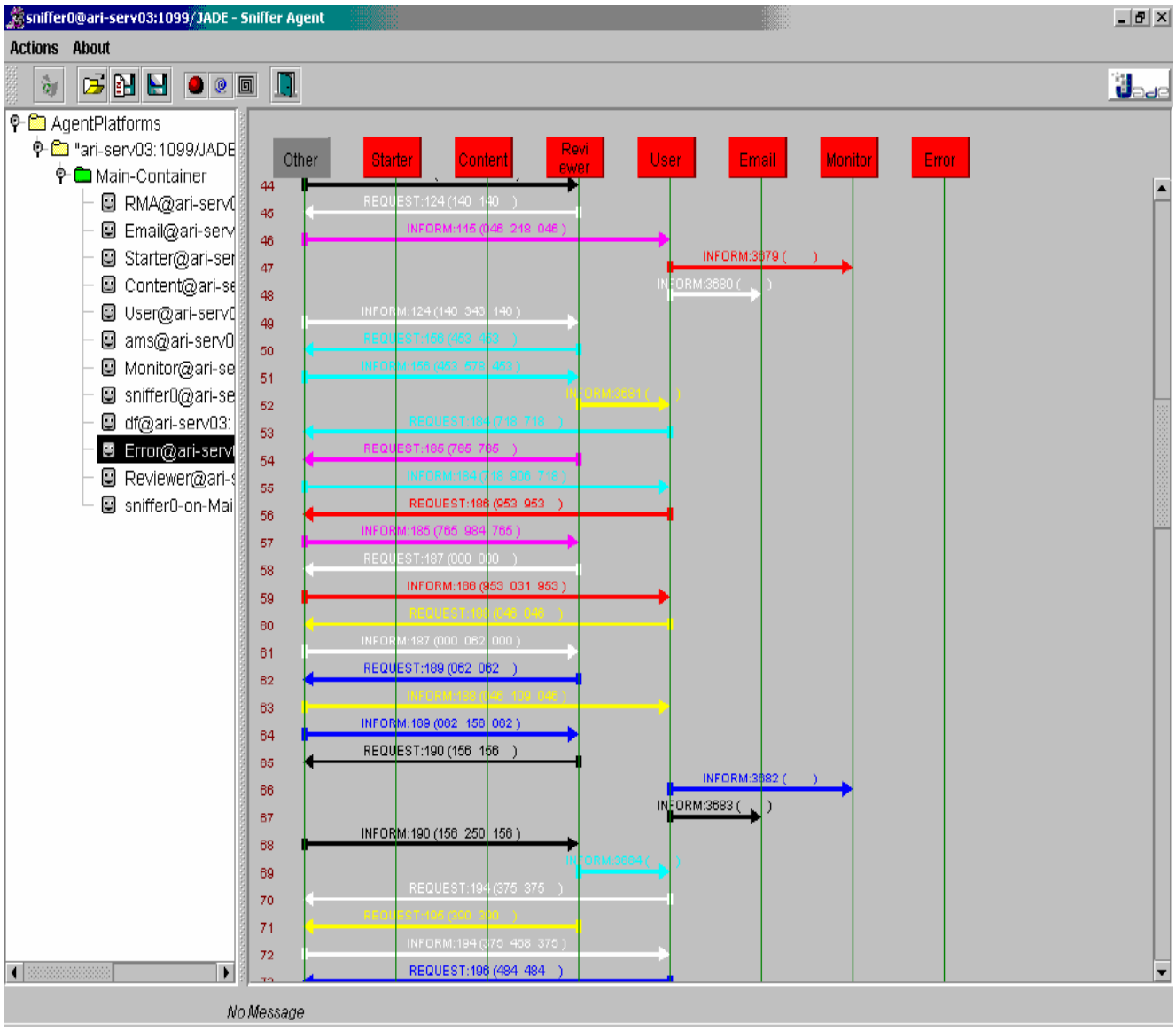

Figure 7: Screenshot showing agent communication

CHAPTER 6

6. Performance Comparison

For the current and the following chapters, we have used the following terms:-

Application – suggests the complete DLNET application. A part of this J2EE based application; the Content Review Process is under study in this work.

Existing application – The part of DLNET that does the content review. It is the application currently being used by DLNET.

Agent Based Application – The application being proposed in this work. It uses software agents for DLNET Content Review and is based on JADE software agent infrastructure.

This part of my thesis focuses on experimental design including the choice of experimental variables, experimental models and the instruments used to measure the dependent variables.

6.1 Experimental Design and Analysis (Methodology)

Experiments are designed to test or prove if a causal relationship exists between the chosen independent and dependent variables. In this work, we are trying to prove that the software framework used for content review in DLNET affects the performance of the system. This is an experiment to prove the effectiveness of a new software design method. With the background of our readings of [7], [8], [13] and [31], we have based our methodology mainly on the reading [24] for design.

These quasi-experiments are designed to test our hypothesis that the software agent based Content Review Process has better response time, is more maintainable and reusable and so has better performance than the J2EE based existing system.

6.2 Experimental Variables

Independent Variable for these quasi-experiments is the type of system with the levels “old system” and “new system”. The “old system” refers to the existing application and the “new system” refers to the Agent based application. We are trying to highlight that the type of system chosen affects the performance of the system. We have chosen to measure the performance by the dependent variables response time, maintainability, scalability, correctness, reliability and reusability. Through our quasi-experiments and results thereof, we prove that the performance of content review

process is better with the agent based architecture. Extrapolating the hypothesis to apply to the whole application, we propose the new framework for DLNET i.e. a software agent based infrastructure.

6.3 Subject Assignment

For the assignment of the subjects to one of the two types of systems, we have used the matching technique of self twinning type. In matching technique for subject assignment, each subject in one group is twinned with a similar subject in another group. The category of matching technique that we have used here is called the self twinning or within subject or related/matched groups. In within subject assignment, each subject is twinned with themselves. This, in effect, involves repeated measures on the subject for different levels of independent variable.

While designing the experiments and test data (the input or the subjects), we have made groups of varying number of resources and then subjected those groups to both the applications. This is done to nullify the effect of the *type* of resources on the results. Since both the applications are getting the same resources, the results will highlight the differences due to the architecture alone.

The condition for using this kind of assignment is that the subjects should not get altered when subjected to one experiment. In our case, the subject is the test data for the two applications. We are using the same test data for the two applications and also rolling back the test database to pretest condition after execution of each test to create exactly same conditions for the two runs.

For our experiments, we have the existing application that is live and the agent based application has been set up to run in parallel. For the duration of our experiments, we disassociate the existing application with the live database and let it run with our test data and test database, which is a copy of the live database. We have created the test input data to simulate different conditions of finding the content reviewer. For each of our tests we first let one of the applications run the test input and update the database. The logs are created and interpreted. Then the test database is rolled back to its pretest state and the second application is executed with the same test data (the input). This is done to simulate random assignment or we can say to have the same effect as random assignment since it was not possible to let the two systems running in parallel get randomly assigned subjects. The basic purpose of random assignment is to have internal validity in the design by ensuring that our treatment groups are similar to each other prior to the treatment. We are ensuring this by having the same set of data being input to the two applications.

In our experiments, we have taken large sample sizes so that the effect of chance is reduced and they are large enough to cover all the variations.

We observed the sample size the application was exposed to for a month and also had discussions with the administrator of the application about the average number of resources the system could expect at a time. Through these observations and discussions, we concluded that the maximum number of resources expected could go up to 50 at one particular time. We added another 25% and tested the applications for up to 62 resources at a time.

6.4 Dependent Variables

The attributes or the dependent variables we have measured to compare the performance of the two systems are:-

6.4.1 Response Time (a measure of latency) –

The time taken by the application to complete the review process is being measured as the response time. We have logged the response times from both the applications for a variety of load conditions ranging from 5 resources to 62 resources covering the average load conditions to heavy load conditions.

In this work, for our experiments we have chosen a small part of the complete DLNET application to prove that software agents based infrastructure provides better performance than the existing J2EE based application. Since the scope of our thesis is content review process of DLNET, we have compared the response time for the two architectures for the content review process only. For content review, the response time is not a critical performance measure as it does not involve any user interaction. This means that the user is not expecting the response from the content review process of DLNET instantaneously. But when we extrapolate our results to cover the whole application, the response time becomes a very important performance criterion. DLNET involves searching the database for the relevant information for the user. The operations involve a lot of user interaction and for that kind of communication, response time becomes important.

6.4.2 Maintainability –

The average time taken to add a new feature or business logic to the content review process is taken as a measure of maintenance. We have tested the maintainability of both the applications by adding the same feature/ business rule in both the applications and recording the average time it would take for a developer to do so. We have tested it with the help of two software engineers with similar background so as to have an average of the measurements and to try eliminating any bias.

6.4.3 Scalability –

This attribute will test the stability of the system in case of increased load. We have created test data of 62 resources which is more than the expected number of resources DLNET will normally receive at a time. It is important to test the system under heavy load conditions.

With time, the number of resources, reviewers and users that DLNET will handle will increase manifold. To be ready for such growth, it is important that the application is scalable.

6.4.4 Correctness –

The correctness of the results is compared. The reviewers selected by both the applications are compared with manually selected reviewers to arrive at a comparison of the number of errors reported by the two applications.

To propose any change in the infrastructure of the application and for any kind of comparison of any two applications, before considering any performance enhancements, it has to be made sure that the correctness or the validity of the applications is maintained.

6.4.5 Reliability/ Failure Rate –

The applications are left running for a long time continuously and failures are recorded to measure the reliability of the applications. Since it is not possible to generate test data for all kinds of real time situations, we let the two applications run parallel uninterrupted and observed them for a week. These results can then be extrapolated to suggest acceptable reliability estimates.

6.4.6 Reusability –

Any application has to be adaptable to cater to changing requirements and/or additional requirements. The new technologies are being designed to increase software reuse and the ease of reuse. It was therefore important for us to compare the two applications' reusability. The existing application is J2EE based and consists of objects and components. Objects and components are encapsulation of functionalities. They are designed so that they can be reused easily. The main problem when reusing objects in a typical J2EE environment is the integration with other objects of the application in case the object needs to be changed for reuse. Using an agent based architecture helps as agents are more independent of other agents/ objects of the

application. To add a new action/ feature in an agent for reusing it involves change in the agent only.

6.5 Test Plan

For having a performance comparison of the two architectures, we have the existing application that is live and we set up the new application to run in parallel to it.

There were a very limited set of tools available for testing the performance of distributed applications as we have concluded based on our readings in [7], [8], [13] and [31], so we devised our own test plan and testing instruments for the measurement of the quality attributes. These quality attributes or dependent variables are an indication of the performance of the applications. After careful study of the application and the content review process, we manually created different groups of test data to simulate typical situations that we wanted to test.

6.5.1 Measurement Instruments:

In the new agent based application, we have developed a monitor agent that makes the log of the response time for the review process every time the application is run. It employs the checkpoint method by recording the time of various events in the process. To this we have also added functionality to generate the log of the resources selected. The log of resources is used in correctness measurement.

For the existing application, we added a piece of code that will also generate similar logs of the response time and the results of the application. This piece of code has been designed carefully to use similar method of recording response time to eliminate any bias in measurements. This enabled us to have a comparison of the two applications.

6.5.2 Test data:

To have the comparison of the applications in an effective manner, we have generated test data to represent the average load conditions and heavy load conditions. By observing the average load conditions of DLNET, on an average it will receive about 5 resources at a time and the heaviest load could be about 50 resources, so we have selected load conditions ranging from 5 resources to 62 resources.

We have generated a test data of about 100 resources and the results in the form of logs created by both the applications are tabulated to get a comparison of the two applications.

6.5.3 Test Procedure:

After careful study of the sample size and the specifics of the applications, we arrive at four different sets of data to measure the response time, correctness and scalability of the applications for. We disassociate the existing application with the live database. We replicate the database as test database for these experiments. Then we link the existing application with the test database and input the test data and manually start the content review process. Logs are generated for the measurement of response time. Next, we start the agent based application that is linked to another copy of the test database. Again, logs are generated for the response time of the proposed application. This completes one of the four quasi-experiments for measurement of response time. The procedure is repeated for the other sets of data.
For the measurement of correctness, the same procedure is followed and the database updates are carefully studied to check if the applications are correct.
The quasi-experiments for correctness also give a measure of the scalability for varying load conditions, so same procedure is followed to test scalability.

6.6 Test results

6.6.1 Response Time

Number of Resources	Response Time of the existing application (msec)	Response Time of the agent based application (msec)
5	3672	1281
62	29063	12422
10	14313	2500
28	22140	7047

Table 2: Response Time Results

6.6.2 Maintainability

To compare the maintainability of the applications, we had three different people add to the applications and then compared the average time taken by them. To have an unbiased comparison, we selected two other persons who had similar experience and understanding of java (the basis of the existing application) and JADE (the agent architecture of the new application).
We added a few features and made some modifications to the two applications.

For these quasi-experiments, we had two other software engineers understand the two applications and make changes to them. We then took the average time taken by them to add to/ modify the two systems.

The following are the results of the maintenance quasi-experiments. For the first three activities, we have considered only two readings as the third person was already familiar with the applications when we started these experiments.

Test condition/ feature added	Time taken to change/ review the existing application			Time taken to change/ review the agent based application		
	Seema	Gaurav	Vaishali	Seema	Gaurav	Vaishali
Understand the working of the application		19.5 hours	20.7 hours		25 hours	26 hours
Understand how to start the application		6.3 hours	5.9 hours		4.5 hours	3.9 hours
Changing the configuration parameters		8 hours	9 hours		2.5 hours	3.1 hours
Adding the creation of log reports	11 hours	13.3 hours	12.9 hours	3.5 hours	4.3 hours	5.4 hours
Identifying a piece of code and replacing with a more efficient one	10 hours	11 hours	9.6 hours	8 hours	8.7 hours	8.3 hours

Table 3: Raw Maintainability Results

The results shown below are the average of the above readings.

Test Condition/ Feature added	**Time taken to change/ review the existing application**	**Time taken to change/ review the Agent based application**
Understand the working of the application	20.1 hours	25.5 hours
Understand how to start the application	6.1 hours	4.2 hours
Adding the feature to change the configuration parameters	8.5 hours	2.8 hours
Adding the creation of log reports	12.4 hours	4.4 hours
Identifying a piece of code and replacing with a more efficient one	10.2 hours	8.5 hours

Table 4: Summary of Maintainability Results

6.6.3 Scalability

We added about 100 resources in one time and then ran both the applications to see if they can both handle the load.

Number of resources added	The existing application could handle the load	The agent based application could handle the load
5 Resources	Yes	Yes
10 Resources	Yes	Yes
28 Resources	Yes	Yes
62 Resources	Yes	Yes

Table 5: Scalability Results

6.6.4 Correctness

For the test data of over 100 resources that we added, we manually selected the reviewers to check if the reviewers selected by the applications were correct.

Number of resources added	Correctness % for the existing application	Correctness % for the agent based application
5 Resources	100	100
62 Resources	100	100
10 Resources	100	100
28 Resources	100	100

Table 6: Correctness Results

6.6.5 Reliability

The two applications were left running for week uninterrupted.

Time for which applications were left running	Number of failures for the existing application	Number of failures for the agent based application
1 hour	0	0
1 day	0	0
1 week	0	0

Table 7: Reliability tests

6.6.6 Reusability

To measure the reusability of the applications, we had two persons reuse a part of the applications and then compared the average time taken by them. Here again, as in maintenance tests, we selected the other person who had similar experience and understanding of java (the basis of the existing application) and JADE (the agent architecture of the new application) as the author.

The following are the results of the reusability quasi-experiments. The identified pieces of code were reused in the applications for additional function. The applications were then tested thoroughly.

Details of Code piece reused	**Average Time taken to reuse and test the existing application (Hours)**			**Average Time taken to reuse and test the Agent based application (Hours)**		
	Seema	Sarala	Pankaj	Seema	Sarala	Pankaj
Code to find suitable reviewers reused as code that finds suitable users and suggests users as reviewers	5.5	7.0	6.3	5.0	6.0	3.9
Code that looks for the presence of any content to be reviewed is reused as code that looks for any inactive reviewers in the database.	5.5	7.5	3.7	4.5	4.3	5.0
Code that logs the names of the selected reviewers is reused as code to log the title and description of the resource.	5.2	4.9	5.1	4.7	4.9	2.6
Code that validates the resource by checking its metadata is reused as code to check the metadata of the user who submitted the resource.	5.4	6.2	4.7	4.6	6.0	5.5

Table 8: Raw Reusability Results

Details of Code piece reused	**Average Time taken to reuse and test the existing application (Hours)**	**Average Time taken to reuse and test the Agent based application (Hours)**
Code to find suitable reviewers reused as code that finds suitable users and suggests users as reviewers	6.26	4.96
Code that looks for the presence of any content to be reviewed is reused as code that looks for any inactive reviewers in the database.	5.56	4.6
Code that logs the names of the selected reviewers is reused as code to log the title and description of the resource.	5.06	4.6
Code that validates the resource by checking its metadata is reused as code to check the metadata of the user who submitted the resource.	5.43	5.36

Table 9: Summary of Reusability Results

CHAPTER 7

7. Performance Comparison Results

Based on the data measured by the testing tools designed as described in the previous chapter, this chapter discusses the results and compares the performance of the two applications.

7.1 Response Time –

Goal: To measure the response times of the two applications with varying load so as to compare them.

Question: Is the Software agent based application faster than the existing application and is this performance constant with varying loads?

Metric method: We designed measuring instruments to create a log of the response times of both the applications. For the existing application, we added a code sample to create the log of the response times each time we conduct the test and for the agent based application, we added another agent to create similar logs.
To eliminate the possibility of bias, we first considered having one code module for both the applications. The two applications have very different architectures and connecting them both to the same module was not possible, so we designed two different modules. We designed them to make sure that we record the time in similar fashion in the two applications. We have designed the instruments to record the response time for both the applications in a similar method. We record the system time before the first line of code in the review process and record the time after the last line of the process. The difference in the two times gives the response time of the applications.

Conclusion: The results show considerable improvements in the response time for the process with the proposed application using Jade infrastructure and software agents as compared to the existing application. The huge improvement in response time can be partly explained by the fact the agent based systems are inherently multithreaded while the existing content review process is a single threaded serial application. We conducted tests for varying load conditions and the results were found to be in the favor of the agent based application. Following is the plot of the results showing a comparison.

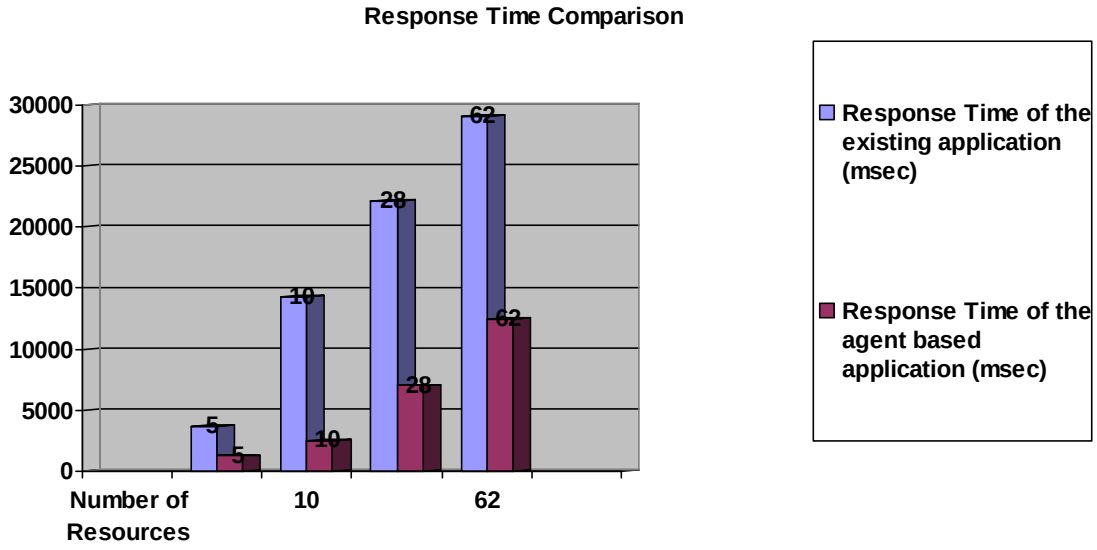

Figure 8: Bar Graph showing a comparison of response times

7.2 Maintainability –

Goal: To design instruments to measure the maintainability of the two applications.

Question: Is the Software agent based application easier to add to / modify than the existing application?

Metric method: To measure the maintainability of the applications, we identified five areas that were put to test. We had two engineers with similar backgrounds who helped us in these experiments. We started with observing the time it takes someone having prior knowledge of java to understand the working of the two applications. Then we noted the time it takes to start the application, then the time it takes to manage the applications like changing certain administrative parameters. We then measured the time it took the same engineers to add a functionality to the two applications. The fifth measure of maintainability was the time it took to optimize a part of the code by changing it and testing the change.

To attempt to eliminate the possibility of bias in these instruments, we carefully selected the people to help us out in this part of the work having similar backgrounds. They were both equally conversant with java but agent based technology was new to both of them

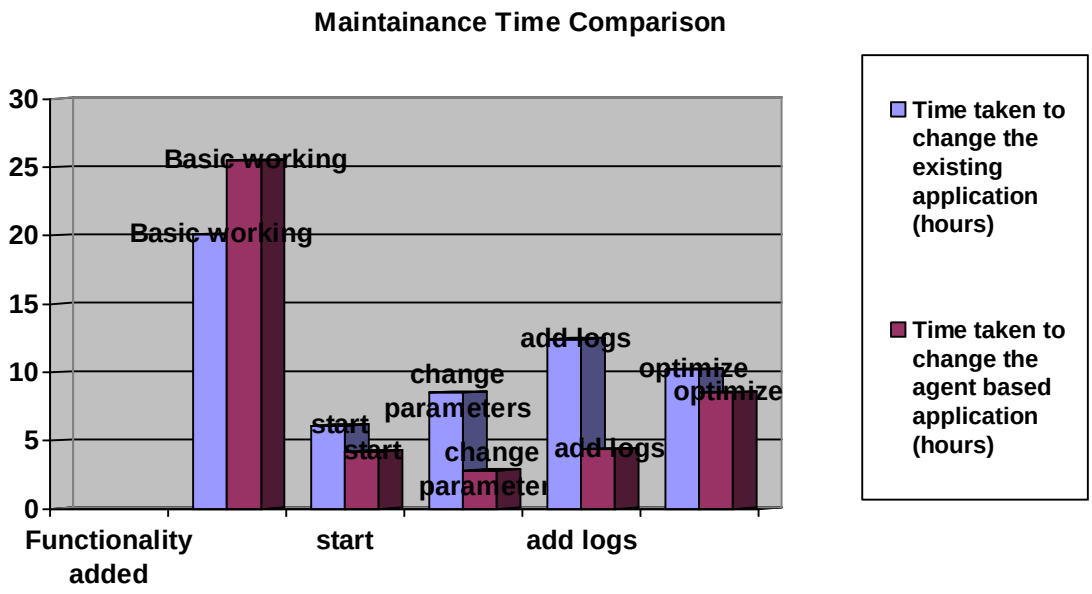

Figure 9: Bar Graph showing a comparison of maintainability

Conclusion: The agent architecture provides a more maintainable architecture as we can see from the results of our tests. The agent based application has modular architecture with agents encapsulating an abstraction. This makes this architecture easy to understand and maintain. As was found out in our measurements, it is much easier for a developer to change the functionality in the agent based application as compared to the existing setup.

The Agent based architecture provides an in built mechanism to graphically view the interactions of various agents in real time. This feature makes it easy to debug these applications. There is no such mechanism in the existing application and in case of any logical errors; the application would mean walking through the code.

The ease of deployment of the two applications can also be compared. The FIPA compliant software agent based application offers standard instructions for loading of the agent container which is very well documented and easy to follow. To get the application agents running, a new user will have to follow instructions given in the manual in appendix B. On the other hand, the existing application is part of the complete DLNET application and therefore needs application specific instructions which can be hard to comprehend.

7.3 Scalability –

Goal: To design instruments to measure the scalability of the two applications.

Question: Is the Software agent based application intrinsically better to handle large loads.

Metric method: To measure the scalability of the applications, we exposed the applications to varying loads. With observation and discussion to the administrator of the application, it was concluded that the maximum load could be about 50 resources at a time. We tested the applications with number of resources ranging form 5 to 62.

Conclusion: Both the applications were found to be equally scalable for the tests conducted.

7.4 Correctness –

Goal: To design instruments to measure the correctness of the two applications.

Question: Of the two applications being studied, does any one give more correct results than the other?

Metric method: The two applications are being tested for different load conditions and we have added modules in both the applications to generate logs of the resources being selected, which is the output of the applications. The modules that were added to log the response time of the two applications were enhanced to log the output (the selected resources) as well.

We maintained the same system status for both the applications for each run of the test and the same resources were being input to the two applications. The effort to restore the system status for both the applications was done to eliminate any possibility of bias in the measurement instruments.
The two applications are supposed to optimally select the best resources for content review based on the current status or the data of the application. For each test, we carefully studied the system status and the input to find what the correct output should be.

Conclusion: Both the applications were found to be correct in all the tests conducted for differing load conditions. So there is no clear advantage of one type of architecture as far as correctness is concerned.

7.5 Reliability

Goal: To test and compare the reliability of the two applications.

Question: Is there any difference in the two applications when comparing their reliability?

Metric method: To measure the reliability of the applications, we let the two applications run in parallel. The applications were observed for any failures when left running for a week uninterrupted.

Conclusion: Both the applications were found to be very reliable in the tests conducted. So there is no clear advantage of one type of architecture as far as reliability is concerned.

7.6 Reusability

Goal: To design instruments to measure the reusability of the two applications.

Question: Does the Software agent based application provide easier reuse of code?

Metric method: To measure the reusability of the applications, we identified two pieces of code in both the applications that could be reused and noted down the time it took to reuse the pieces of code elsewhere in the application.
The time also included the time it took to test the application after integrating the new pieces of code.

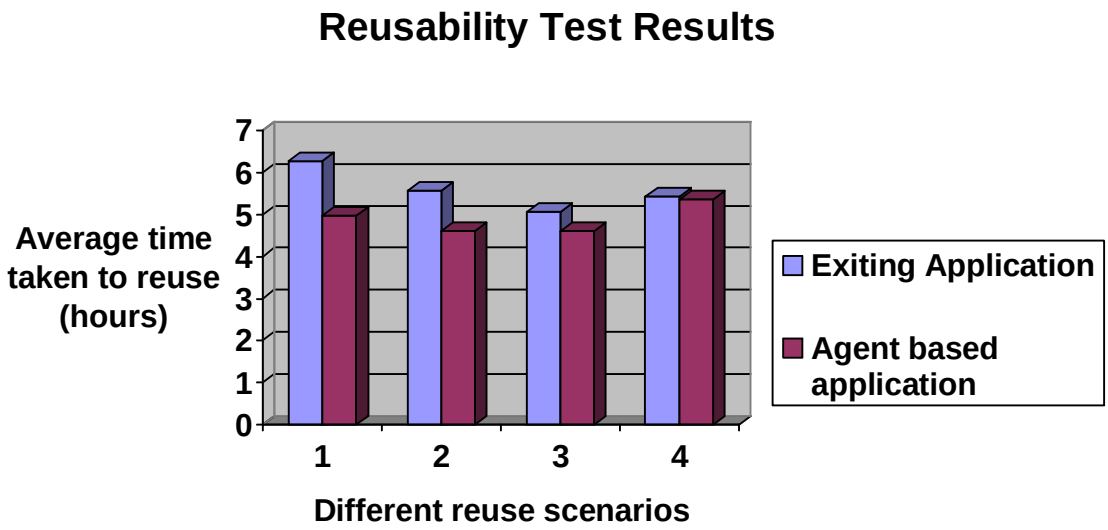

Figure 10: Bar Graph showing Reusability Results

Conclusion: The agent architecture provides a more reusable architecture as we can see from the results of our tests. Reusability is related to maintainability and the results indicate that the modular structure of agent based system makes it more reusable in addition to making it more maintainable.

7.7 Discussion of results

Looking at the different test results, we see that both applications score equally well on scalability and correctness, although the software agent based application was found to be much faster. The response time was found to be much better for all the load conditions in the proposed system. The reason for this mainly lies in the fact that agent based system are multithreaded where the existing application is more a serial application. This reflects an inherently faster architecture; we believe it'll be a very useful parameter as DLNET becomes more popular and with the increase in the frequency and the number of resources submitted.

The maintainability results indicate that although the initial understanding of the agent based application took more time than the understanding of the existing application, the subsequent changes to agent based application are much faster than the existing application. We believe that in the current era when the applications are so dynamic in nature, it is useful to have an application that can adapt to changes fast.

The agent based system was found to be much more maintainable as it has its elements modularized as agents and it is easier to understand the interactions and the functionality encapsulated in each software agent.

The JADE infrastructure of the software agent based application provides a graphical user interface to monitor the interactions between the various agents and that can be helpful in understanding and monitoring the system. The existing java based application provides no such feature. The proposed application is therefore more usable. The ease of changing the configuration parameters in the agent based application is also a very helpful feature for the user/ administrator.

Being modular and autonomous in design, the agents based application is more reusable than the existing application as is discussed in section 6.4. The interdependence of code in the existing application makes it difficult to test it once a change has been made to reuse a part of the code.

Having discussed these performance test results, we would also like to mention that these tests were done on a specific part of the DLNET application (the content review portion) and with simulated test data. Although we have tried to design our tests to be generic so as to reflect the true performance of the application, the actual results when comparing the whole application or another part of the application might be different.

We would also like to mention that we had tried to make the new application replicate the original application but some parts of the existing application were found to be

non functional at the time of the tests, like the email module that should send email messages to the reviewers was not working. We created the logs of the text messages that would have been sent as email messages instead.

CHAPTER 8

8. Conclusions and Future Work

8.1 Conclusions

The contributions outlined in chapter 1 are met by the design of the agent based application for the content review for DLNET and by the quasi-experiments to compare the performance of the same with the existing Content Review Process application. This thesis gave exploratory evidence that the concept that software agent based architecture will make the current content review portion of the DLNET application more efficient, maintainable and reusable. There are various variables which were measured to compare the two applications' performance, and this comparison is the main contribution of this thesis. We measured the response time, maintainability, scalability, correctness, reliability and reusability of the two applications to quantify the performance of the two technologies, namely J2EE versus Agent Technology. While response time, correctness and reliability indicate the effectiveness of the applications at this time, maintainability, scalability and reusability on the other hand, reflect the ease of further updates, enhancements and modifications.

From our study we can conclude that agent architectures can be used in digital library applications such as DLNET where a great need for interoperability, rapid application development and automation exists. This study started with these benefits in mind and we were able to give sufficient evidence that further work in this direction will be very useful. We therefore recommend future research work in the areas listed in section 8.2. Other possibilities such as streamlining of the authentication and licensing procedures, security of the application and improvement in search facilities can also be explored.

The proposed application

- has fully automated the content review process of DLNET.
- has provided easy way to change configuration parameters.
- improved the response time of the application by 57% to 82%.
- improved the maintenance time of the application by 16% to 67%.
- improved the reusability by 1% to 26%.

These results can be further generalized beyond the current study and sample, by following these procedures for other modules of DLNET.

As a result of these quasi-experiments and the study, we concluded that the proposed content review process using software agents is a good strategy for the redesign of DLNET.

8.2 Open Issues

We would like to highlight that in this thesis work, we had planned to have the two applications run exactly the same but some modules of the existing application were found to be not working like the classes that are used to send email messages to reviewers once the reviewers have been selected. We had to conduct our experiments by generating logs of such messages. It would have been better if we could really send the messages and really put the applications to use.
We have proved by our experiments that the proposed application will be faster (better response time) and easier to maintain but some work needs to be done to make this application live and really useful. What we have done is simulation of the real life load conditions. Actual utility by putting the application to real life situations remains to be worked upon.

8.3 Future work

We have implemented an agent based content review process for DLNET in this work but this is just the beginning. It can be considered as a start of a series of enhancements to the existing system not only limited to content review but the whole DLNET application.
We propose DLNET to have a design like the architecture of Next Generation Digital Library where we have the J2EE infrastructure to provide the basic services of the library and the use of software agents to administer the advanced facilities and features.
Following is a list of additions that can be investigated to be implemented with software agents to improve the current system:-

8.3.1 Content/ Reviewer Harvesting

The agents can be used to look for the resources available on the internet. There are a lot of fields in our taxonomy where we lack the resources in our library. The application can be enhanced to look for content in such areas.
We can extend our application to have agents that can contact the academia for students and professors informing about our library. They can then choose to become members and contribute to the growth of DLNET.

8.3.2 Increase Membership

Agents can be used to advertise the Digital Library to university students and academia professionals to encourage them to use the library. The use of the library will only give us insight into what is required to make it all the more user friendly.

8.3.3 Optimize Reviewer Selection

The reviewer selection in the current setup can be further optimized by first getting more information about the users when they register as reviewers and then this data can be used when our process is selecting the reviewers.

References

1. Information Agents: A new Challenge for AI *Daphne Koller and Yoav Shoham*, Stanford University –
   This work provides a stimulus for using software agents, where they are most applicable etc.

2. Web Economics: A case for Agent Based Digital Libraries *Innes A. Ferguson, Jorg Muller, Markus Pischel and Michael Woldridge.*
   This work presents the ZunoDL and how agents are being used to build digital libraries.

3. Agent-Based Digital Libraries: Driving the Information Economy. *David Derbyshire, Innes A. Ferguson, Jorg P. Muller, Markus Pischel and Michael Wooldridge*

4. The university of Michigan Digital Library Service Market Society *Jose M.Vidal, Tracy Mullen, Peter Weinstein, Edmund H.Durfee,* University of Michigan.
   The architecture of agent based digital library of University of Michigan is presented in this paper.

5. The agent architecture of the University of Michigan Digital Library *E.H. Durfee, D.L.Kiskis, W.P.Birmingham*
   Another paper describing the agent based digital library at University of Michigan.

6. Paying their way: Commercial Digital Libraries for 21st century – *Innes A. Ferguson and Michael J.Wooldridge, Zuno Ltd.*
   Also at http://www.dlib.org/dlib/june97/zuno/06ferguson.html

7. Comparison of Two Component Frameworks: The FIPA-Compliant Multi-Agent System and The Web-Centric J2EE Platform – *Michelle Casagni, Margaret Lyell*

8. Performance Testing of Software Systems – *Filippos I.Vokolos, Elaine J. Weyuker*

9. http://jade.cselt.it/doc/JADEProgramming-tutorial-for-beginners.pdf
   A manual for beginners of Jade agent architecture.

10. Technical aspect of Next Generation Digital Library Project: *Hiroshi Mukaiyama*
    The architecture of the Next Generation Digital Library and reasons of adoption of the architecture are described here.

11. A Java-based Smart Object Model for use in Digital Learning Environments – *Vara Prashanth Pushpagiri*
    A report on the Learning Object Model of DLNET.

12. Next Generation Digital Library – Architecture and Implementation – *Hiroshi Mukaiyama*
    Another paper on the agent based digital library highlighting the benefits of using agents.

13. Early Performance Testing of Distributed Software Applications –*Giovanni Denaro, Andrea Polini, Wolfgang Emmerich* - Suggests ways to test the performance of a distributed application from the design phase itself.

14. Jade A White Paper – *F.Bellifemine, G.Caire, A.Poggi, G.Rimassa*
    An introductory paper on Jade agent architecture.

15. http://jade.cselt.it/
    The official website of jade agent architecture from where it may be downloaded and there are various links to numerous related resources.

16. Tools and Techniques for Performance Measurement of Large Distributed Multiagent Systems – *Aaron Helsinger, Richard Lazarus, William Wright, John Zinky*

17. http://www.dlib.org/metrics/public/papers/The_Dlib_Test_Suite_and_Metrics.pdf - Gives an idea about collecting software metrics to compare the performance attributes.

18. Assembling and Enriching Digital Library Collections – *David Brainbridge, John Thompson and Ian H. Witten*
    This paper highlights the content collection and processing for Digital Libraries in particular for Greenspan Digital Library

19. An Agent-Based Hypermedia Framework for Designing and Developing Digital Libraries – *Michail Salampasis, John Tait and Colin Hardy*
    This paper presents another agent based model for University of Sunderland Digital Library

20. Automatic Document Metadata Extraction using Support Vector Machines – *Hui Han C. Lee Giles, Eren Manavoglu, Hongyuan Zha*
    This literature presents some ideas about automatic metadata extraction from content.

21. Research Issues for Digital Libraries - *David A. Forsyth*
    This paper focuses on some research issues associated with digital libraries of which particularly of our interest were metadata generation.

22. Developing a Digital Library Of Computer Science Teaching Resources - *Scott Grissom, Deborah Knox, Elana Copperman, Janet Hartman, Marja Kuittinen, David Mutchier, Nick Pariante*
    This paper presents some ideas for content collection and review for digital libraries.

23. Agent-based Information Retrieval – *http://www.cs.umbc.edu/abir/*
This article enhances our understanding about agent for use in information retrieval applications.

24. Software Metrics : A Rigorous Approach – *Norman E Fenton*
A book which guided the experiment design and analysis of this work.

25. Analytical Usability Evaluation for Digital Libraries: A Case Study – *Ann Blandford, Suzette Keith, Iain Connell & Helen Edwards.*
This study analysis various techniques for comparing the usability of digital libraries.

26. Strategic Directions in Electronic Commerce and Digital Libraries: Tpwards a Digital Agora – *Nabil Adam and Yelena Yesha et al.*
A survey paper comparing electronic commerce and the digital libraries concepts.

27. Data Mining Challenges for Digital Libraries – *Robert L. Grossman*
A paper highlighting the future challenges for digital libraries.

28. AGS: Introducing Agents as Services Provided by Digital Libraries – J.Alfredo Sanchez, John J. Leggett, John L. Schnese.

29. http://www.dlnet.vt.edu

30. An approach to Performance Evaluation of Software Architectures – *Simonetta Balsamo, Paola Inverardi and Calogero Mangano*

31. Capacity and Performance Analysis of Computer Systems – *James N. Robinson*

32. Cougaar Architecture Document – *A BBN Technologies Document*

33. Jade Programmers' Guide – *Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa*

34. Jade Administrator's Guide - *Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa*

35. Analysing Internet Software Retrieval Systems: Modelling and Performance Comparison – *Jose Merseguer, Javier Campos and Eduardo Mena*

36. JADE A FIPA2000 Compliant Agent Development Environment – *Fabio Bellifemine, Agostino Poggi, Giovanni Rimassa*

37. An Agent Based Approach to the Construction of Floristic Digital Libraries - *J.Alfredo Sanchez, Cristina A.Lopez, John L. Schanase.*

38. A Model for Virtual Intelligent Libraries – *Guadalupe Munoz Martin*

39. http://www.isogenic.info/html/13__sample_size.html
40. http://www.son.wisc.edu/rdsu/library/Sample%20size.pdf

Appendix A - Definitions/ Terms used

DLNET

The digital library network for Engineering and Technology is a project funded by National Science Digital Library (NSDL) initiative of National Science Foundation. It is a collaborative effort of four institutions, namely, the American Society for Engineering Education, the Institute of Electrical and Electronics Engineers, Inc., Iowa State University and Virginia Tech. It is a J2EE based application developed and maintained with the support of National Science Foundation. DLNET at http://www.dlnet.vt.edu is envisaged as a platform for information discovery, interaction, content-building and distribution that will support pedagogy and learning in Engineering and Technology. It was launched in September 2000 and DLNET got operational in 2003. [29]

Software Agent

A software agent is a high level abstraction that significantly simplifies the process of building complex systems, is defined in terms of its behavior. It is characterized by being autonomous, communicative with users and other agents and being perceptive to surroundings.

Learning Resource

A structured electronic resource that encapsulates high quality information in order to facilitate learning and pedagogy has a stated objective and a targeted audience [11]. Note: The protocol is not to start a sentence with a reference

Independent/ Dependent Variables

As per [24], an experiment usually involves the application of well crafted and carefully controlled manipulations to some subject of study, followed by a series of measurements that have been selected or developed to characterize the effects of different manipulations on the subject. In the language of experimentation, each manipulation that is applied to the subject of study is called an independent variable and the outcome measure used to characterize the effects of the independent variable in the subject is called the dependent variable.

JADE

Jade is an enabling technology, a middleware for the development and run-time execution of peer-to-peer applications which are based on the agents paradigm and which can seamlessly work and interoperate both in wired and wireless environment [14].

Reliability

"Reliability is the probability of a device performing its purpose adequately for the period of time intended under the operating conditions encountered." – [Billington and Allen (1983)] Note: Be consistent in the use of references. Recommend you use [ ].
"Software Reliability is defined as the probability of failure free operation of a computer program in a specified environment for a specified time" – [Musa and Ianino (1987)]

Error

A discrepancy between a computed, observed or measured value or condition and the true, specified, or theoretically correct value or condition.

Failure

The termination of the ability of a functional unit to perform its required function. [ANSI/IEEE Std 729-1983]

Peer to Peer

Peer to peer computing model is the type in which all the participating nodes have equal capabilities. Another distinguishing factor is that any node can initiate communication as opposed to a client server model in which only the client can initiate communication with the server.

Appendix B - Manual of the new application module implementing Content Review Process

Introduction

The proposed application using software agents is based on FIPA (Foundation for Intelligent Physical Agents) based agent architecture named JADE (Java Agent Development Environment). The official website of jade is http://jade.cselt.it. It is a java based infrastructure and new agents are developed in java language. The infrastructure provides a library of classes that the agents can use, a runtime environment for the agents and also graphical tools for administering the applications.

Additions/ Modifications

New agents can be added as per the following procedure:-

1. Identify the need for a new agent and the behaviors it needs to have.
2. Write the Agent class.
3. The agent has to be registered in the Service Directory.
4. The behaviors need to be added.
5. The communication with other agents has to be streamlined.
6. The messages to other agents can then be added in whatever sequence as per the desired requirement.

With an example we'll show how to add another agent to the application. In the current application, we have a few agents and suppose we want to add another agent to the system. Say we want to add another agent that optimizes the current selection of reviewers. With reference to chapter 5 describing the current application, the Reviewer and the User agents would like to communicate with this new agent.

Developing the new agent, AgentOptimizer, we want to first register the agent so that other agents that want to communicate with it can find this new agent. This is done as follows:-

```
protected void setup()
  {
        ServiceDescription sd  = new ServiceDescription();
    sd.setType( "optimizer" );
    sd.setName( getLocalName() );
    register( sd );
   }
        void register( ServiceDescription sd)
        {
```

```
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        dfd.addServices(sd);
        try {
                DFService.register(this, dfd );
        }
        catch (FIPAException fe) { fe.printStackTrace(); }
}
```

On the other side, the calling agent will first look for the agent of the type “optimizer”

```
// trying to look for an agent with the optimizer type of service
    DFAgentDescription dfd1 = new DFAgentDescription();
    ServiceDescription sd1  = new ServiceDescription();
    sd1.setType( " optimizer" );
    dfd1.addServices(sd1);
    DFAgentDescription[] result = DFService.search( myAgent, dfd1);

        AID ag_optimizer = new AID();
        ag_optimizer = result[0].getName();
```

And then send appropriate message to initiate action.

```
ACLMessage snd_msg1 = new ACLMessage(ACLMessage.INFORM);
snd_msg1.setContent(“OPTIMIZE”);
snd_msg1.addReceiver( ag_optimizer);
send(snd_msg1);
```

Adding a behavior

New requirements or enhancements can be added to the agents as additional behaviors. The code for the behavior of an agent goes in the action method of the agent class file.
Since agents only respond to some messages they receive from other agents, we first have to receive the message and then take appropriate action based on the message we receive.

```
public void action()
        {
// Initialize local variables
    int count=0;
    String msg;
// Receive the message
ACLMessage rcv_msg= receive();

if (rcv_msg!=null)
```

```
                    {
// Take appropriate action based on the message content

                    if (AgentStarter.STARTTIME.equals(msg))
                    {
                            // Do whatever needs to be done.
                            // The task details

                    } // end of if (AgentStarter.STARTTIME.equals(msg))
```

Appendix C– Sample Logs Created

Log created by the agent based application lists the reviewers selected

Today is Jun 29, 2004 Resources found = 16
Resource = DLNET-06-28-2002-0477 Reviewer found = danieleewww
Resource = DLNET-06-25-2004-0138 Reviewer found = seemamitra
Resource = DLNET-06-27-2004-0141 Reviewer found = sreviewer3
Resource = DLNET-06-27-2004-0142 Reviewer found = sreviewer4
Resource = DLNET-06-27-2004-0143 Reviewer found = sreviewer
Resource = DLNET-06-28-2004-0144 Reviewer found = sreviewer3
Resource = DLNET-06-28-2004-0145 Reviewer found = hanoudi
Resource = DLNET-06-28-2004-0226 Reviewer found = sreviewer
Resource = DLNET-06-28-2004-0227 Reviewer found = sreviewer2
Resource = DLNET-06-28-2004-0229 Reviewer found = sreviewer4
Resource = DLNET-06-28-2004-0228 Reviewer found = sreviewer3
Resource = DLNET-06-28-2004-0230 Reviewer not found
Resource = DLNET-06-28-2004-0230 Guest Reviewer found = Vara Prashanth
Resource = DLNET-06-27-2004-0139 Reviewer found = seemamitra
Resource = DLNET-06-27-2004-0140 Reviewer found = sreviewer2
Resource = DLNET-06-28-2004-0231 Reviewer not found
Resource = DLNET-06-28-2004-0231 Guest Reviewer found = Vara Prashanth
Resource = DLNET-06-29-2004-0146 Reviewer found = sreviewer3

Logs created by agent based application to log the response time of the application

Today is Jul 3, 2004 Time Review Started = 1088893359406
Time Guest Reviewer Found for one resource = 1088893360593
Time Guest Reviewer Found for one resource = 1088893360843
Time Guest Reviewer Found for one resource = 1088893361093
Time Guest Reviewer Found for one resource = 1088893361328
Time Reviewer Found for one resource = 1088893361421
Time Reviewer Found for one resource = 1088893361609
Time Guest Reviewer Found for one resource = 1088893361906
Time Guest Reviewer Found for one resource = 1088893362125
Time Guest Reviewer Found for one resource = 1088893362375
Time Reviewer Found for one resource = 1088893362484
Time Reviewer Found for one resource = 1088893362656
Time Reviewer Found for one resource = 1088893362828
Time Reviewer Found for one resource = 1088893363015

Logs created by the existing application to log the reviewers selected and the time taken by the application

Today is Jun 29, 2004   Time Review Started = 1088522158343
Resource = DLNET-06-28-2002-0477
Reviewer Found = danieleewww
Resource = DLNET-06-25-2004-0138
Reviewer Found = seemamitra
Resource = DLNET-06-27-2004-0141
Reviewer Found = sreviewer3
Resource = DLNET-06-27-2004-0142
Reviewer Found = sreviewer4
Resource = DLNET-06-27-2004-0143
Reviewer Found = sreviewer
Resource = DLNET-06-28-2004-0144
Reviewer Found = sreviewer3
Resource = DLNET-06-28-2004-0145
Reviewer Found = hanoudi
Resource = DLNET-06-28-2004-0226
Reviewer Found = sreviewer
Resource = DLNET-06-28-2004-0227
Reviewer Found = sreviewer2
Resource = DLNET-06-28-2004-0229
Reviewer Found = sreviewer4
Resource = DLNET-06-28-2004-0228
Reviewer Found = Vara Prashanth
Resource = DLNET-06-28-2004-0230
Reviewer Found = Vara Prashanth
Resource = DLNET-06-27-2004-0139
Reviewer Found = seemamitra
Resource = DLNET-06-27-2004-0140
Reviewer Found = sreviewer2
Resource = DLNET-06-28-2004-0231
Reviewer Found = Vara Prashanth
Resource = DLNET-06-29-2004-0146
Reviewer Found = Vara Prashanth
Time Review Process Ended = 1088522165265

Appendix D– Sample Code of the Agents created

```
package dlnet_jade;

import jade.core.Agent;
import jade.core.AID;
import jade.core.behaviours.*;
// To add Directory Service Facility
// The DF associates service descriptions to Agent IDs (AID)
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;

//import jade.domain.AMSService;
//import jade.domain.FIPAAgentManagement.*;

import jade.lang.acl.*;

public class AgentStarter extends Agent
{

// These are all the environment variables and other constants used in the application

	public final static String SEARCHREVIEWER = "SEARCHREVIEWER";
	public final static String SEARCHGUESTREVIEWER =
"SEARCHGUESTREVIEWER";
	public final static String REVIEWERNOTFOUND =
"REVIEWERNOTFOUND";
	public final static String EMAILREVIEWER = "EMAILREVIEWER";
	public final static String EMAILGUESTREVIEWER =
"EMAILGUESTREVIEWER";
	public final static String STARTREVIEW = "STARTREVIEW";
	public final static String EMAILMESSAGE = "You have a resource to review.";

	public final static String REVIEWSTARTED = "REVIEW";
	public final static String REVIEWSTART = "RVWSTART";
//	public final static String REVIEWSTART = "REVIEW";

	public final static String STARTTIME = "STARTTIME";
	public final static String REVFOUND = "REVFOUND";
	public final static String GUESTFOUND = "GUESTFOUND";
	public final static String NOREVFOUND = "NOREVFOUND";

// Environment Variables
```

```
        public final static int   NUM_REV_TO_FIND =                 1;
        public final static int   MAX_REVIEW_LOAD =                 3;
        public final static int   MAX_RECENT_REVIEW_LOAD =          2;

        // Location of documentation

        public final static String        DL_docBase =
        "c:/dlnet/" ;
        // Location of content storage

        public final static String        DL_LO_Record_Location  = new
String(DL_docBase +
                                          "repository/tempStorage/contentStorage/") ;
        // Manifest Location
        public final static String        authorEmailSS  =       new String(DL_docBase +

        "systemFiles/manifestDisplay/getAuthorEmail.xsl" );
        // User Profile Location
        public final static String        DL_UserProfile_Loc  =         new
String(DL_docBase +
                                                  "repository/userProfiles/" ) ;



   protected void setup()
   {
                ServiceDescription sd  = new ServiceDescription();
      sd.setType( "starter" );
      sd.setName( getLocalName() );
      register( sd );

// Send messages to the Agent Content to start looking for content to be reviewed
        // Send this message every day

                addBehaviour(new MyCyclicBehaviour(this));
}

                class MyCyclicBehaviour extends CyclicBehaviour
                {
                        public MyCyclicBehaviour(Agent a)
                        {
                                super(a);
                        }
                        public void action()
                        {
```

```
                        AID ag_mon = new AID();

      try {

// trying to look for an agent with the content type of service
         DFAgentDescription dfd = new DFAgentDescription();
         ServiceDescription sd1  = new ServiceDescription();
         sd1.setType( "content" );
         dfd.addServices(sd1);

         DFAgentDescription[] result = DFService.search( myAgent, dfd);
//          System.out.println(result.length + " results" );
         if (result.length>0)
         {
 //             System.out.println("The searched agent is  " ); // + result[0].getName() );

                              ACLMessage msg = new
ACLMessage(ACLMessage.INFORM);
                              msg.setContent( STARTREVIEW );
                              AID dest = result[0].getName();
                                    msg.addReceiver( dest);
                              // Send the message to a Content Agent
                              System.out.println("Sending message to COntent Agent
\n");
                              send(msg);
                        }
                        // trying to look for an agent with the monitor type of service
                              DFAgentDescription dfd1 = new DFAgentDescription();

                              ServiceDescription sd2  = new ServiceDescription();
                              sd2.setType( "monitor" );
                              dfd1.addServices(sd2);
System.out.println("Looking for monitor");
                              DFAgentDescription[] result1 = DFService.search(
myAgent, dfd1);

                               if (result1.length>0)
                              {
                                 System.out.println("The searched Monitor agent is  " );
//+ result[0].getName() );

                                                      ag_mon = result1[0].getName();
                                                      ACLMessage msg = new
ACLMessage(ACLMessage.INFORM);
                                                      msg.setContent( STARTTIME );
                                                      msg.addReceiver( ag_mon);
```

```
                                                    // Send the message to the monitor
Agent
                                                    System.out.println("Sending
message to Monitor Agent \n");
                                                    send(msg);

                                                }

        }
        catch (FIPAException fe)
        {
                        fe.printStackTrace();
                }

                                //block(1500000000) ;
                                try
                                {
                                        Thread.sleep(500000);
                                }
                                catch(Exception e)
                                {}
                                // This interval can be set to whatever frequency we want
                                // the content search done

                        }

          }

          void register( ServiceDescription sd)
                {
                        DFAgentDescription dfd = new DFAgentDescription();
                        dfd.setName(getAID());
                        dfd.addServices(sd);

                        try {
                                DFService.register(this, dfd );
                        }
                        catch (FIPAException fe) { fe.printStackTrace(); }
                }

}
```

Vitae

Seema Mitra nee Aggarwal, daughter of late Sh. Bal Chandra Agrawal and Kusum Aggarwal, was born on September 15, 1971 in New Delhi, India. She earned her Bachelor's degree in Instrumentation and Control Engineering from Delhi Institute of Technology, New Delhi in August 1992. Having worked on a variety of engineering jobs like consultancy and software development in leading organizations viz. Engineers India Ltd., Infosys Technologies Ltd. and NIIT Ltd. over a period of 9 years, she got back to academics in Spring of 2003. Since Jan 2003 she has been involved in pursuance of Masters in Computer Science at Virginia Tech. After finishing her graduate school, she wants to get back to research oriented work in the field of Software Engineering and Design.