A multi-agent based architecture for Digital Libraries

C. Georgousopoulos

A Ph.D. dissertation submitted to the School of
Computer Science of the University of Wales, Cardiff

Supervisor: Dr. O. F. Rana

2005
Cardiff, U.K.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Dedicated to my loving father and mother,

Nikolaos & Maria

and to my 'second' mom *Efthimia Ntavantzi*

A Multi-Agent based architecture for Digital Libraries

Georgousopoulos Christos

Abstract

Digital Libraries (DL) generally contain a collection of independently maintained data sets, in different formats, which may be queried by geographically dispersed users. The general problem of managing such large digital data archives is particularly challenging when the system must cope with data which is processed on demand. This dissertation proposes a Multi-Agent System (MAS) architecture for the utilisation of an active DL that provides computing services in addition to data-retrieval services, so that users can initiate computing jobs on remote supercomputers for processing, mining, and filtering of the data in the library. The system architecture is based on a collaborative set of agents, where each agent undertakes a pre-defined role, and is responsible for offering a particular type of service. The integration of services is based on a user defined query which can range in complexity from simple queries, to specialised algorithms which are transmitted to image processing archives as mobile agents. The proposed architecture enables new information sources and services to be integrated into the system dynamically, supports autonomous and dynamic on-demand data processing based on collaboration between agents, capable of handling a large number of concurrent users. Focus is based on the management of mobile agents which roam through the servers that constitute the DL to serve user queries. A new load balancing scheme is proposed for managing agent load among the available servers, based on the system state information and predictions about lifetime of agent tasks and server status. The system architecture is further extended by defining a gateway to provide interoperability with other heterogeneous agent-based systems. Interoperability in this sense enables agents from different types of platforms to communicate between themselves and use services provided by other systems. The novelty of the proposed gateway approach lies in the ability to adapt an existing legacy system for use with the agent-based approach (and one that adheres to FIPA standards). A prototype has been developed as a proof-of-concept to outline the principles and ideas involved, with reference to the Synthetic Aperture Radar Atlas (SARA) DL composed of multi-spectral remote-sensing imagery of the Earth. Although, the work presented in this dissertation has been evaluated in the context of SARA DL, the proposed techniques suggest useful guidelines that may be employed by other active archival systems.

Acknowledgment

Although this is one of the first pages of this dissertation it was the last one to be written. Eventually, I have come to the end of a long "journey" full of experiences and memories. During my research I have faced a lot of difficulties but due to the help of certain people I have succeeded to reach the end of my work of the last five years.

I would like to thank my supervisor, Dr. Omer F. Rana, without whom I would have not been able to finish my research. I truly appreciate his advice and inspiration on my thesis. His valuable guidance, excellence advice, constructive feedback on my work and insight brought me to the end of one of the most important things on my life, hopefully the acquisition of my Ph.D. degree. I also would like to express my thanks to Professor W. A. Gray who gave me the opportunity to start my research and prove to myself that I CAN do it.

Thanks to my family; my father Nikolaos, my mother Maria, and my two sisters Stavrilena and Christina-Maria. I would not have been able to survive writing this thesis without their support and understanding. Their encouragement and help gave me strength to accomplish my objective. Many thanks to Dr. Yanyan Yang, a prior computer associate of Cardiff University for her valuable advice and contribution on my research; Dr. Holger - a core member of the FLASH project and Mr. Christian Erfurth for providing me with information on their related research. I am also thankful to Mr. Peter Cameron and Miss Helen Pickavance from ObjectSpace Inc. for their technical support on Voyager in correlation with Java; Mr. Sven Kaffile, my college Dr. Steven Lynden and Mr. Alastair Duncan (a FIPA-OS team member) for their assistance on FIPA-related programming. I am also grateful to Dr. Antonys Karageorgou, a core member of the Manchester node of AgentCities, for his assistance on testing the interoperability of the FIPA-compliant gateways, and Mr. Robert Evans - the system manager and assistant director of laboratories in the department of computer science of Cardiff University, for his excellent assistance on technical aspects.

Finally, I would like to thank my good friends for helping and encouraging me for the last five years, my brother-in-law Papaevagelou Ioannis (the "president"), Kokonas Christodoulos and Golegos George. Last but not least, I would like to express my gratitude to Mr. Taki and Mrs. Niki Oikonomou for their support.

Publications directly relevant to this dissertation
(in chronological order)

2000

- **Mobile Agents and the SARA Digital Library**

  Yang Y., Rana O. F., Georgousopoulos C., Walker D. W., Williams R. D.,

  a) In proceedings of IEEE Advances in Digital Libraries 2000, held in Washington, D.C., pages 71-77. Published by the IEEE Computer Society Press, ISBN 0-7695-0659-3, 22-24 May 2000.

  b) CARC - Center for Advanced Computing Research. Technical Report CARC-186, 2000.

- **Agent Based Data Analysis for the SARA Digital Library**

  Rana O. F., Yang Y., Georgousopoulos C., Walker D. W., Williams R. D.,

  In proceedings of the International Workshop on Advanced Data Storage/Management for High Performance Computing, held at CLRC-Daresbury Laboratory, Warrington, U.K., pp 211-220. Available as Daresbury Laboratory Technical Report DL-CONF-00-001. ISSN 1362-0223, 23-25 February 2000.

- **A Multi-Agent System for Analysing Synthetic Aperture Radar Atlas (SARA) Data**

  Rana O. F., Yang Y., Georgousopoulos C., Walker D. W., Williams R. D.,

  Project Report published in AgentLink (Europe's ESPRIT-funded Network of Excellence for agent-based computing), Issue 5, pp. 11-13, ISSN 1465-3842, May 2000.

2002

- **Agent based data management in Digital Libraries Remote-Sensing Archive**

  Yang Y., Rana O. F., Walker D. W., Georgousopoulos C., Aloisio G., Williams R. D.,

  Published in Parallel Computing Journal, Elsevier Science, vol. 28, issue 5, pp. 773-792, 2002.

- **An approach to conforming a MAS to a FIPA-compliant system**

  Georgousopoulos C., Rana O. F.,

  a) In First International Joint Conference on Autonomous Agents and Multi-Agent Systems - AAMAS 2002, ACM ISBN 1-58113-480-0, Italy, Bologna, pp. 968-975, 2002

  b) Presented in UKMAS 2002 - UK Workshop on Multiagent Systems, 18 & 19 December, 2002.

- **Towards an XML and Agent-Based Framework for the Distributed Management and Analysis of Multi-Spectral Data**

  Rana O. F., Yang Y., Georgousopoulos C., Walker D. W., Aloisio G., Williams R. D.

  In Proceedings of the 6th International Digital Media Symposium on Intelligent Agents for Mobile and Virtual Media, held 23-26 April 2001 at Bradford , UK. Springer-Verlag, ISBN:1-85233-556-4, pp. 77-88, 2002.

2003

- **Combining State and Model-based Approaches for Mobile Agent Load Balancing**

  Georgousopoulos C., Rana O. F.,

  In SAC 2003 - ACM Symposium on Applied Computing, ACM ISBN 1-58113-624-2, Melbourne, Florida, USA, pp. 878-885, March 2003.

2004

- **Supporting FIPA Interoperability for Legacy Multi-Agent Systems**

  Georgousopoulos C., Rana O. F., Karageorgos A.,

  In Agent Oriented Software Engineering (AOSE) Workshop of Autonomous Agents and Multi-Agent Systems (AAMAS03) conference, held in Melbourne, Australia, July 2003. Published in Lecture Notes in Computer Science series: Agent-Oriented Software Engineering IV, ISBN: 3-540-20826-7, 2004.

2005

- **Performance-sensitive Service Provision in Active Digital Libraries**

  Georgousopoulos C., Rana O.F.,

  In International Conference on e-Technology, e-Commerce and e-Services - IEEE'05, held in Hong Kong, March 29-April 1 2005. Published by IEEE Computer Society, ISBN 0-7695-2274-2, 2005.

- **An agent infrastructure for on-demand processing remote-sensing archive**

  Yang Y., Rana O.F., Walker D.W., Williams R. D., Georgousopoulos C., Caffaro M., Aloisio G.,

  In Journal of Digital Libraries, Springer Verlag, Issue vol.5, Number 2, pp. 120-132, ISSN 1432-5012, 2005.

- **Mobile Agent-based Service Provision in Distributed Data Archives**

  Georgousopoulos C., Rana O.F.,

  In Scientific Applications of Grid Computing: First International Workshop, SAG 2004, Beijing, China, September 20-24, 2004. Published by Lecture Notes of Computer Science, Springer Verlang, vol. 3458, ISSN 0302-9743, ISBN 3-540-25810-8, 2005.

Table of Contents

List of figures

List of tables

List of code-segments

List of formulas

Glossary of acronyms

ACC - Agent Communication Channel
ACL - Agent Communication Language
ADL - Alexandria Digital Library
AI - Artificial Intelligence
AID - Agent IDentifier
AMS - Agent Management System
AMT - Agent Message Transport
AP - Agent Platform
API - Application Program Interface
BDI - Beliefs Desires Intentions
CASBA - Common Agent Service Brokering Architecture
CORBA - Common Object Broker Architecture
CGI - Common Gateway Interface
DAI - Distributed Artificial Intelligence
DB - Data-Base
DBMS - Data-Base Management System
DCD - Document Content Definition
DF - Directory Facilitator
DL - Digital Library
DPS - Distributed Problem Solving
DPSS - Distributed-Parallel Storage System
EXSA - External Service Agent
FD - Federated Directory
FIPA - Foundation of Intelligent Physical Agents
FLASH - Flexible Agent System for Heterogeneous Cluster
GA - Gateway Agent
GIS - Geographical Information System
GUI - Graphical User Interface
HPSS - High Performance Storage System
HSM - Hierarchical Storage Management
HTML - Hyper Text Mark-up Language
HTTP - Hyper Text Transport Protocol
IDL - Interface Definition Language
IIOP - Internet Interoperable ORB Protocol
IP - Internet Protocol

ISP - Internet Service Provider
JADE - Java Agent Development Framework
JAI - Java Advanced Imaging
JAS - Java Agent Services
JDBC - Java Data-Base Connectivity
JDK - Java Development Kit
JPL - Jet Propulsion Laboratory
JSDK - Java Servlet Development Kit
KIF - Knowledge Interchange Format
KQML - Knowledge Query Meta Language
LAA - Local Assistant Agent
LB - Load Balance
LADE - Local Averaging-algorithm Dimension-Exchange
LADF - Local Averaging-algorithm DiFfusion
LIA - Local Interface Agent
LIGA - Local InterGration Agent
LMA - Local Management Agent
LRA - Local Retrieval Agent
LSA - Local Security Agent
MA - Management Agent
MAS - Multi-Agent System
MASIF - Mobile Agent System Interoperability Facility
MATS - Mobile Agent Team System
MPI - Message Passing Interface
MTP - Message Transport Protocol
MTS - Message Transport Service
NASA - National Aeronautics and Space Administration
NFS - Network File System
NS - Naming Service
NTP - Network Time Protocol
OCEAN - Open Computation Exchange & Auctioning (or Arbitration) Network
ODBC - Open Data-Base Connectivity
OMA - Object Management Architecture
OMG - Object Management Group
ORB - Object Request Broker
P2P - Peer-to-peer
PVM - Parallel Virtual Machine

RAID - Redundant Array of Independent Disks
RDF - Resource Description Framework
RFI - Request For Information
RMI - Remote Method Invocation
RPC - Remote Procedure Call
RSI - Recursion Software Inc
SAR - Synthetic Aperture Radar
SARA - Synthetic Aperture Radar Atlas
SDSC - San Diego Supercomputer Center
SGML - Standard Generalised Mark-up Language
SI - System Integration
SQL - Structured Query Language
SSL - Secure Sockets Layer
TCP - Transmission Control Protocol
UAA - User Assistant Agent
UIA - User Interface Agent
UIUC - University of Illinois at Urbana-Champaign
UMA - Universal Management Agent
UMDL - University of Michigan Digital Library
UMIST - University of Manchester Institute of Science and Technology
URA - User Request Agent
URAS - URA's Servant
URL - Uniform Resource Locator
VCL - Virtual Community Library
VM - Virtual Machine
VPN - Virtual Private Network
WAP - Wireless Application Protocol
WWW - World Wide Web
XML - eXtensible Markup Language
XSL - eXtensible Style-sheet Language

Chapter 1. Introduction

A Digital Library (DL) is a vast collection of objects stored and maintained by multiple information sources, including databases, image banks, file systems, email systems, the Web, and other methods and formats. Digital libraries involve the management, analysis, integration and annotation of large data sets, maintained on various platforms, and managed by different administrators. The data sets can also vary in complexity and type, with repositories storing image data, sound and video samples, and textual data. Hence, digital libraries can enable data from multiple sources to be integrated in intelligent ways, generally to support the discovery of new scientific insights by collectively analysing data from different scientific domains.

1.1. Motivation

The amount of digital spatial data available is growing rapidly. In particular, there is a vast amount of data from Earth observation satellites. This presents a challenge for the development of software systems to enable the storage, management and dissemination of these huge datasets in on-line data archives or digital libraries. Ideally, such a system should provide efficient, on-demand remote access to these datasets over the Internet, so that authorised users can easily access and utilise the data for a variety of applications including geology, image registration, resource monitoring etc. For a number of spatial applications, such as satellite imagery, the processing requires high-performance compute servers. In addition, scientists often require integrated access to information combining retrieval, computation, and visualisation of individual or multiple datasets. Scientific collaborations are already distributed across continents, and software to enable these work groups will become increasingly vital. It will be necessary for human interfaces to these archives to become more simple to use and flexible. This has led to the concept of an *active* digital library [32][161], where users can process available data not just to retrieve a particular piece of information, but to infer new knowledge about the data at hand. The term "active" implies that the library provides computing services in addition to data-retrieval services, so that users can initiate computing jobs on remote supercomputers for processing, mining, and filtering of the data in the library. In the scientific world, scientists need to deal with both data-centric and process-centric views of information. While it is important to have access to information, often it is also important to know how the information was derived. Hence, the scientist should have a technological infrastructure

that can intelligently and automatically process the distributed data, thereby transforming the processed data into useful knowledge.

The general problem of managing such large digital data archives is particularly challenging when the system must cope with data which is processed on demand. Active data is data that is dynamically generated by a scientific experiment, or it may be obtained from a sensor or monitoring instrument - known as remote-sensing data. Remote-sensing data about the Earth's environment is being created at an ever-increasing rate and distributed among heterogeneous remote sites. Such remote-sensing image data is often useless without a sophisticated, customisable data-mining and knowledge extraction process. Knowledge mining extracts information from the large data set, and the wide distribution of data at multiple sites often requires an intelligent fusion of the data from multiple space agencies. Among many different paradigms and architectures of distributed computing systems for a remote-sensing archive [33][88][162][163] the mobile agent paradigm appears to be the most promising solution.

1.2. Approach and results

This dissertation proposes a Multi-Agent System (MAS) architecture utilising an active digital library composed of multi-spectral remote-sensing imagery of the Earth, as part of the Synthetic Aperture Radar Atlas (SARA) which is referred to as SARA architecture within this thesis. The existing SARA DL maintains a data repository of 40 TB in total, acquired by the SIR-C shuttle in 1994/95. Although the original data set is small compared to other high performance computing applications, the resulting analysis on images can lead to large quantities of data, some of which must be integrated with data from other systems, such as Geographical Information Systems (GIS[1]) or data gathered from ground stations.

The system architecture comprises a number of collaborating agents, where each agent undertakes a pre-defined role and is responsible for offering a particular type of service. The integration of services is based on a user defined query which can range in complexity from simple queries, to specialised algorithms which are transmitted to image processing archives as mobile agents. The functionality required for on-demand processing of remote-sensing archives can be decomposed into different classes

[1] *A GIS is a computer system capable of assembling, storing, manipulating, and displaying geographically referenced information. A common use of a GIS is to overlay several types of maps (e.g. train routes, street maps) to determine useful data about a given geographic area.*

of agents to achieve the desired goals. The SARA architecture enables new information sources and services to be integrated into the system dynamically, supports autonomous and dynamic on-demand data processing based on agents' collaboration, capable of handling a large number of concurrent users.

The SARA agent-based architecture is extended by defining a gateway to provide interoperability with other heterogeneous agent-based systems; interoperability, in the sense that agents from different types of platforms can communicate between themselves and use services provided by other system. The interoperable gateway of the digital library conforms to the FIPA (Foundation of Intelligent Physical Agents) standard; the IEEE Computer Society has formally accepted FIPA to become part of its family of standards committees in 2005[54]. In this instance, information on the digital library may be further enhanced by the integration of data retrieved from a FIPA-compliant system (i.e. that adheres to FIPA specifications), such as a GIS capable of interoperating with the digital library. The longitude and latitude of a particular area of the Earth can be used as parameters within a GIS to retrieve land information such as street names, which can then be combined with SARA image(s) of the corresponding geographical coordinates, resulting in a detailed map of the particular area.

In addition, the scalability of the architecture utilising the SARA active DL is further optimised by the introduction of management agents, responsible for improving the mobile agents' itinerary and balancing the load of their tasks within the MAS. The proposed technique of load balancing, which is based on a combination of state-based and model-based approaches of LB, apart from ensuring a coherent distribution of agents among the servers, also enables the realisation of a monitoring system and provides caching techniques based on similarity identification of prior agent requests.

The development of a prototype provides the basis of evaluation for the multi-agent architecture of the SARA active DL. Experimental results demonstrate the successful achievement of System Integration and Data Management within the agent-based architecture of collaborative agents proposed for the utilisation of the DL, the ability of the system to interoperate with external FIPA-complaint agent-based systems by utilising the FIPA-compliant gateways, as well as the even distribution of agent load among the servers that constitute the DL. Further experiments have also been conducted to test the adaptability of the load balancing model, with positive results.

Note that the implementation of SARA prototype has been developed within the department of Computer Science of the University of Wales, Cardiff. Apart from the experimental tests conducted on the interoperability of the architecture (utilising the FIPA-compliant gateways approach) in collaboration with UMIST (University of Manchester Institute of Science and Technology)[2] University, the rest of experimentation has been based within the department. The test cluster was connected to the University network. In order to minimize the likelihood of network traffic influencing the performance of experimental tests, experiments were performed at early hours in the morning, after midnight.

The key contributions of this dissertation are in the context of active digital libraries and agent technology, in the area of agent interoperability and load balancing. More specifically:

- The dissertation presents a complete, secure agent-based architecture for the realisation of an active digital library which is both modular and extensible. The flexibility of the proposed architecture lies in the ability to transfer custom analysis algorithms to resource servers for local data fusion and analysis, and the integration of information and services provided by external agent-based systems.
- Under the attempt to define an interoperable gateway for the proposed architecture of the digital library, an alternative approach to provide FIPA-compliance to an agent-based legacy system has been proposed. The novelty lies in the ability to conform a legacy multi-agent system to an interoperable one - i.e. one that adheres to FIPA standards. This is achieved with the use of gateways which behave like wrappers between the non-FIPA compliant system and a FIPA-compliant one. In this instance, any legacy agent-based system may utilise the gateways approach to adopt FIPA compliance. The development of an API (Application Program Interface) for the realisation of the gateways enables the extension of the architecture to provide support for more complex interaction of heterogeneous agents, not initially supported by the default architecture.
- The distribution of the agent load among the resource servers that constitute the digital library is based on a novel load balancing scheme that combines the most attractive features of existing load balancing approaches. Decisions on load balancing are not based only on system state information (as in a common state-based approach) but on estimations of the lifetime of agent

tasks and predictions of utilisation of servers (as in model-based approaches). In addition, the adaptability of the model that calculates the itinerary of mobile agents is enhanced by an algorithm that has been invented to overcome situations where the prediction of agent tasks tends to be erroneous. The model of the proposed load balancing scheme is generic and may be easily amended for other systems operating data archives.

1.3. Organisation

The remainder of this dissertation is organised in nine chapters as follows. Chapter 2 describes the SARA DL, the deficiencies of the existing system and, briefly, the proposed approach to extend the capabilities and resolve the deficiencies of the current system. Chapter 3 provides the required infrastructure, terminology, concepts and definitions of the three main areas on which this dissertation is focused on i.e. Digital libraries, interoperability of multi-agent systems and load balancing of mobile agents within a MAS, as well as the background work that has been done in those fields. Chapter 4,5 and 6 are the most important chapters where the agent-based architecture of the SARA active DL, the interoperability of the proposed architecture and the management of the mobile agents within the MAS with reference to load balancing are presented. Implementation issues relating to the prototype developed for the SARA active digital library are presented in Chapter 7, whereas Chapter 8 provides a demonstration of experiments conducted on the SARA prototype. Chapter 9 suggests further work that remains to be done and which may provide the motivation for new research studies. Finally, the dissertation concludes in Chapter 10. A glossary of acronyms used within the dissertation is after the table of contents.

Chapter 2. Problem statement

2.1. Introduction

This thesis proposes an agent-based architecture for the utilisation of an active digital library, with reference to SARA - a digital library of multi-spectral remote sensing imagery of the earth. This chapter provides a description of the SARA DL, the deficiencies of the existing system and the proposed approach to extend the capabilities and resolve the deficiencies of the current system.

2.2. SARA digital library

The Synthetic Aperture Radar Atlas (SARA) is a digital library of multi-spectral remote sensing imagery of the earth[4], 40 TB in total, acquired by the SIR-C shuttle in 1994/95, which provides web-based on-line access to a library of data objects at Caltech, the San Diego Supercomputer Center (SDSC) and the University of Leece in Italy[160][161]. Although the original data set is small compared to other high performance computing applications, the resulting analysis of images can lead to large quantities of data, some of which must be integrated with data from other systems, such as Geographical Information Systems (GIS) or data gathered from ground stations. The data is maintained in different kinds of file systems, such as Sun NFS (Network File System), IBM/Livermore HPSS (High Performance Storage System), and delivered using web front-ends. The web interfaces act as an integration tool for combining different server implementations.

The opening web-page presents a user with a picture of the Earth, illustrated in Figure 2.1, and clicking on it the user can zoom in or out, depending on how the rest of the form is set. More distant views show coasts, countries, and rivers; closer views show roads, railways, and city names. Rectangular tracks on the surface of the map indicate that a SAR image (or a collection of more than one) is associated with that region. A user may select a particular track from the map by clicking on it. Alternatively, a user can create a polygon over the surface of the map, from which latitude and longitude coordinates are derived. If multiple tracks contain the chosen point then the user is asked to select from a list of those tracks.

When a track has been selected, a Java applet appears, with a thumbnail image of the track area together with other controls that allow selection of a subset of the full track, an output format, and the false-

coloring information. The processed multi-spectral data (images) may be further processed by choosing a mapping from the frequency/polarisation channels to be red, green and blue components of the final image. This mapping may be optimised to highlight aspects such as ground ecology or snow/ice conditions.

2.2.1. The data objects of SARA

The data maintained within the SARA system was acquired by the space shuttle, during a week-long flight, covering an area of roughly 50 million square kilometers. The data was acquired using a SAR - Synthetic Aperture *active* Radar, which measures the strength and round-trip time of the microwave signals that are emitted by a radar antenna and reflected off a distant surface object. Hence, each pixel in the generated image corresponds to radar backscatter. Darker areas in the image represent low backscatter, and bright areas represent high backscatter. The amount of backscatter depends on the size of the scattering objects in the target area, the moisture content of the target area, the polarisation of the pulses, and the observation angles. The microwave transmissions are vertical and horizontal polarisation combinations: HH (horizontally transmitted, horizontally received), VV (vertically transmitted,

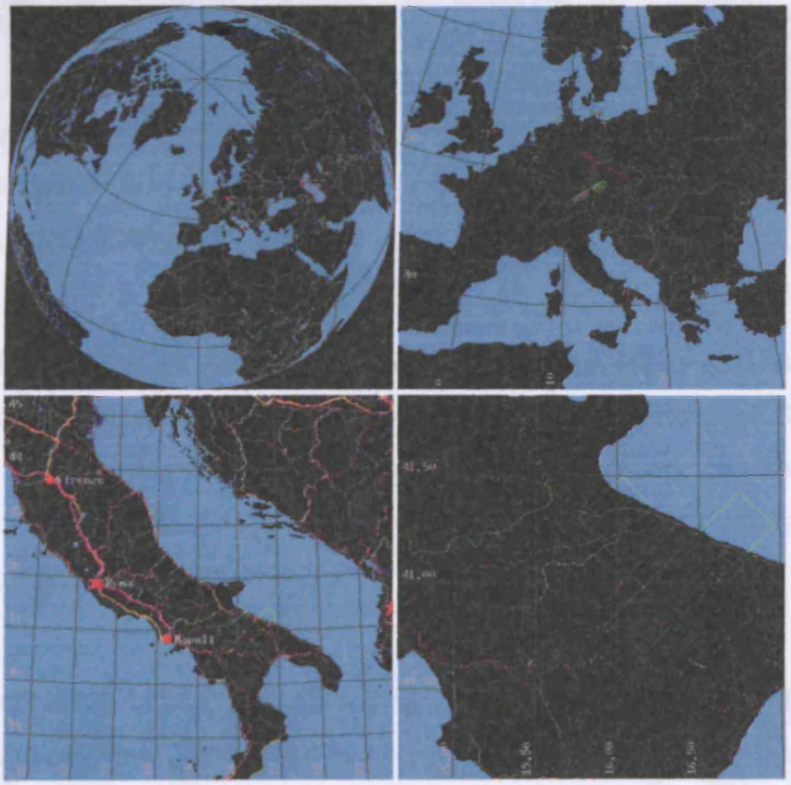


Figure 2.1. SARA map of the globe; zooming in, in an Italian region[129]

vertically received) and VH and HV. This enables the derivation of the complete scattering matrix of a scene on a pixel by pixel basis. Subsequent analysis involves allocating colors to these polarisations to identify particular surface features, such as vegetation cover and sub-surface discontinuities. Additional details of how the imaging radar works can be found at [150].

SAR is an important source of high-volume remote sensing data. This is because SAR[176] can see through clouds, vegetation, and sometimes even a few meters of sand. It can provide imagery of the ground in all-weather conditions at all times. SAR images are used in many fields. SAR has been used to see deep enough into sandy deserts to discover a lost ancient city on the Silk Road, and can identify eco-friendly farming taking place beneath the canopy of the Amazon rain forest. SAR can measure the moisture content of Kansas cornfields, and differentiate spruce from birch in the Russian taiga. SAR can trace the movement of Chilean glaciers, document the destruction of African gorilla habitat, probe the geology of Hawaiian volcanoes, determine the vintage of Antarctic sea-ice, and monitor the recovery of Yellowstone from forest fires.

2.3. Deficiencies of the existing SARA system

The current SARA system[5] provides an interface to a library of data objects, and it allows users to define exactly the subset of the data they want, and retrieve it from any server that has it. However, in contemplating an extension to a system that can handle complex, supervised processing and data-mining, the existing architecture seems deficient.

Firstly, a user usually needs to get familiar with the query and process mechanisms in the system, then s/he must just formulate an appropriate query and wait for the query to complete. The approach not only overloads the user but also may incur inefficiency and delay in query processing, especially when information sources are slow or unavailable, and when significant processing is required for the translation, filtering, mining and merging steps.

Secondly, the present SARA architecture is based on stateless Common Gateway Interface (CGI) scripts so that each request stands alone, rather than in a context of previous requests. It is difficult for the CGI script to create multiple data objects in response to a request, and the output cannot be flushed until the

whole data object has been completed. As a CGI script is independent of the web servers, every user request will start a new CGI progress, which may easily lead to performance bottlenecks with an increase in user requests. Furthermore, error handling in the CGI scripts is also not robust.

Thirdly, such remote-sensing image data is often useless without a sophisticated, customisable data-mining and knowledge extraction process. Knowledge mining extracts information from the large data sets, and the wide distribution of data at multiple sites often requires an intelligent fusion of the data from multiple space agencies. Hence, it is unrealistic to deliver the large volume of scientific data over the internet.

2.4. The proposed solution to the problem

The approach proposed in this thesis for the utilisation of the SARA digital library based on agent technology provides a promising solution to the deficiencies of the existing system and suggests useful guidelines that go beyond the SARA system.

The agent-based infrastructure that has been developed for on-demand processing of remote sensing archives comprises a number of collaborating agents, where each agent undertakes a pre-defined role, such as a user assistant agent, a database query agent, a query-migration agent etc. Each agent is responsible for offering a particular type of service, and the integration of services is based on a user defined query. Queries can range in complexity from SQL (Structured Query Language) queries, to specialised algorithms which are transmitted to image processing archives as mobile agents. The functionality required for on-demand processing of remote-sensing archives can be decomposed into different classes of agents to achieve the desired goals. The most complex functionality is localised in stationary agents, which remain at one location, providing resources and facilities to lightweight mobile agents that require less processor time to be serialised, and are quicker to transmit. The mobile agent assumes the existence of some common infrastructure structure on the server side - hosting the data set. The number of repositories is pre-defined in this application, although the itinerary followed by each agent is not. SARA mobile agents are persistent and can wait for resources to become available. Agents allow the delivery and retrieval of data to complete without user monitoring or recovery actions. The SARA architecture enables new information sources and services to be integrated into the system

dynamically, and supports autonomous and dynamic on-demand data processing based on agents' collaboration, capable of handling a large number of concurrent users.

The realisation of the agent-based architecture led to the design and development of an interoperability layer of the system enabling SARA to interoperate with foreign agencies hosting data archives (e.g. agent-based GIS systems), and vice-versa, thus extending the capabilities/services provided to users. For instance, information retrieved from the SARA system can be further enhanced by additional information gathered from a GIS system that is capable of interoperating with SARA. The longitude and latitude of a particular area of the earth can be used as parameters on a GIS to retrieve land information such as street names, which can then be combined with the image based on geographical coordinates in SARA, resulting in a detailed map of the particular area. Likewise, foreign Multi-agent systems can interoperate with SARA and use its information. Of course, due to the use of mobile agents in the system, a mechanism to load the balance of agent tasks was a necessity and therefore focus has also been given to area of load balancing.

2.5. Conclusion

This chapter described the SARA digital library and gave a brief description of the alternative approach proposed in this thesis for utilising the DL based on agent technology which not only provides an answer to the deficiencies of the existing system but also scales the capabilities of the DL. Details on the architectural design of the system can be found on chapters 4,5 and 6.

Chapter 3. Background and related work

3.1. Introduction

This chapter is divided in two main sections (3.2 and 3.3) to cover the theory and related work of this research accordingly. Section 3.2 presents the required infrastructure, terminology, concepts and definitions of the three key areas on which this thesis is focused on with reference to Digital Libraries and agent-technology, efforts on agent standardisation, and management of agents within a MAS with emphasis to load balancing. Section 3.3 provides a review of the most important attempts on the field of digital libraries, interoperability of multi-agent systems and load balancing from which the research on SARA architecture has been influenced.

3.2. Background

The theoretical part of this chapter begins with what a Digital Library is, how and what is needed for its realisation. Agents are introduced as an emerging technology which can provide a solution not only in the field of Digital Libraries but in other areas too, especially due to their autonomous, intelligent, interactive and adaptive but most of all mobile nature. The importance of a means of communication within an agent-based system composed of multiple agents with different roles and characteristics, as well as the ability of agents to be able to interoperate with agents from external system(s), and how this could be achieved is also defined. Finally, the need of agent management with reference to balancing the load of agent tasks within a MAS is described.

3.2.1. Digital Library

A Digital Library (DL) is a vast collection of objects stored and maintained by multiple information sources, including databases, image banks, file systems, email systems, the web, and other methods and formats. Digital libraries can be viewed as infrastructures for supporting the creation of information sources, facilitating the movement of information across global networks, and allowing the effective and efficient interaction among knowledge producers, librarians, and information and knowledge seekers[106].

Usually, the information sources constituting a DL are heterogeneous, in terms of how the objects are stored, organised, managed and the type of platform on which they reside. In addition, the information sources can be characterised as dynamic in the sense they may be added or removed from the DL system. Furthermore, DLs are composite multimedia objects comprising different media components including text, video, images or audio. Research in DLs has therefore generally focused on providing seamless and transparent access to such objects in spite of the heterogeneity and dynamic among the information sources, and the composite multimedia nature of the objects.

To support these requirements, it is important to provide a means to organise objects within a DL to allow multiple heterogeneous data sources to co-exist, and to provide support for managing vast quantities of data representing each digital object. Integrating different information sources generally requires the development of a system wide data model, and subsequent translation of each source specific data model into the system wide model. Therefore, *System Integration* and *Data Management* are major prerequisites for the realisation of a Digital Library, and are both discussed in the following *3.2.1.1* and *3.2.1.2* sub-sections.

3.2.1.1. System Integration

System integration (SI) in digital libraries requires the ability to deal with massive amounts of objects (usually multimedia ones). In general, SI includes pre-integration, identification of schema matching, schema integration, and source-data integration sub-processes[41][118].

The pre-integration process deals with transforming the data models used within the underlying information sources into a common model i.e. a model that can represent all the models at the underlying source to resolve integration problems due to the heterogeneity in data models such as relational, object-oriented, and hierarchical formats. Specific issues to integration in DLs are a direct result of their characteristics. These include:

- storage support for *large quantities of data*, and support for *structured, semi-structured* and *unstructured data*. The underlying models should enable updates to both content and source schema.

- support for *multimedia data sets*, enabling visual (image and video), stream-based (video and audio) and textual (data sets) to be managed and archived. Metadata is needed to support the management of such data sets, either through the automatic extraction of metadata, or via catalogues which express attributes about the data being stored.

- support for *modifying source schema*, such as in the context of scientific computing where new experimental procedures may necessitate a change in the schema for recording the output of an experiment. Various approaches can be adopted, ranging from database triggers, to analysing database logs for identifying when a particular change should be effected.

- support different *user capabilities and needs*, to enable users with different physical, technical, linguistic and domain expertise to access stored objects. For instance, depending on the device from which access is being made, the DL should automatically send data suitable for that device, such as users equipped with a text-only display media accessing a multimedia data store (with text, video, audio capability) should receive only the textual data part of the corresponding source. User preferences and profiles may also be used for identifying requirements, based on past query history of the user.

The most commonly employed techniques for supporting these requirements are discussed in the section below.

3.2.1.1.1. Approaches to SI in digital libraries

One of the main goals in SI is to provide the capability to interoperate with heterogeneous sources. Three of the most popular approaches to resolve heterogeneity between different types of sources are namely, CORBA (Common Object Broker Architecture)[35], mediators and agents. It is important to note that these three approaches are not orthogonal in the sense that a mediator may employ CORBA and an agent may use mediators.

3.2.1.1.1.1. CORBA approach

The Common Object Request Broker Architecture (CORBA), one of the components of Object Management Architecture (OMA) developed by Object Management Group (OMG), came into existence because of lack of programming interfaces and packages that can deal with heterogeneous platforms[152]. The main components of OMA include object services, common facilities, domain interfaces, application interfaces and the Object Request Broker (ORB).

CORBA consists of numerous features, including ORB Core, Interface Definition Language (IDL), Stubs, Skeletons and others. ORB Core is responsible for delivering requests to object implementation and responses from objects to the user requesting the service. The main feature of ORB Core is its abstractions of the object implementation. While requesting for services, the user does not need to know where the object is located, how the object is implemented i.e. which programming language is used, the state of the object and how to communicate with the objects i.e. via TCP (Transmission Control Protocol)/IP, RPC, etc. All the user needs to worry about is their own application and how to specify the objects of interest. Specifying objects of interest is done though object references. IDL generates two components: Stub and Skeleton. The Stub is responsible for creating and issuing user requests, while the Skeleton is responsible for delivering requests to the object implementation. Stub and Skeleton are specific to object implementation.

In regard to integration, object implementation can be used to define interfaces for interacting with the data source. Even though CORBA provides abstraction of the implementation of services at the object implementation (services provided by the data source), the task of integrating multiple data as responses from multiple object implementations must be performed by the user application. Thus the user application needs to know, to some degree, the metadata of the responses of each object implementation. Furthermore, since IDL is specific to object implementation, changes to the services provided by the data source require changes to the object application and propagation of the updated Stub and Skeleton. This leads to a complex and customised user application. The mediated approach, discussed next, attempts to address these issues.

3.2.1.1.1.2. Mediated approach

The mediated approach utilises two components called mediators and wrappers, to perform integration. The function of a wrapper is to interact with its corresponding information source, converting mediator queries represented in the common language into queries native to the source and vice-versa. To perform its task, a wrapper must have the knowledge of the underlying source. The complexity of a wrapper depends on the amount of co-operation from the source itself, for instance the wrapper might need to perform additional processing on the results received from the source before sending them to the mediator.

To help deal with these heterogeneous sources, the CORBA approach can be used. If CORBA is employed, wrappers do not need to deal with the different interfaces of the source, but need to focus only on formatting the response to query into the common format used within the integration components.

The function of a mediator is to accept users' queries and translate them into the common model. Each query can be broken into smaller sub-queries. Subsequently, each query is sent to the appropriate source via a wrapper. Upon receiving results of sub-queries, the mediator combines and integrates these results to form the complete outcome and presents this to a user. Whenever more than one mediator must be stored and maintained in heterogeneous systems, CORBA can also be used to hide the complexity of the different systems.

To perform its task, the mediator must have the knowledge of the source and their schema to determine which source provides what information. This is one of the limitations of the mediated approach, as the number of information sources need to be pre-defined, and it is not possible for a mediator to discover new sources of interest.

3.2.1.1.1.3. Agent-based approach

The agent-based approach comprises a collection of agents, where each agent has local decision capabilities to perform specialised tasks on behalf of a user or another agent. An agent has knowledge about how to perform its specialised task. Agents can interact with a user, with other related agents and with information sources. There are many types of agents such as collaborative, learning, interface,

information etc. Agent-based integration systems in general are comprised of three types of agents: interface, mediator and source agents. Services offered to users can be dynamically added or updated, based on the use of mobile agents (code) that can update the capability of a given information source, providing adaptability and scalability to DLs. Mobile agents that encapsulate executable code may also be dispatched to a remote server hosting large data sets, in scenarios where moving the computation to the data sets is a more realistic and feasible approach, compared to migrating large quantities of data to a central server for analysis.

Mediator agents interact with interface agents, source agents and other mediator agents. There are many types of mediator agents in which each type performs specific intermediate tasks, including accepting user queries, evaluating user profiles if any, locating the appropriate source agents based on user queries, sending queries to appropriate source agents, monitoring query progress, formatting and integrating responses from source agents, and communicating and working together with other mediator agents to accomplish a task.

An ontology can be used to resolve heterogeneity in terms and definitions used among the agents, as it defines the working model of entities and interactions in some particular domain of knowledge or practices, such as image analysis. In Artificial Intelligence (AI) according to T.Gruber an ontology is "the specification of conceptualisation, used to help programs and humans share knowledge"[75]. In order to send a query to the appropriate agents, a repository of agent description and services is maintained. To locate desired services, the agent can consult the repository. Alternatively, each agent may have the capability to describe their services and to send their description to other agents in a way that can be understood by other agents. To send a query for processing, a mediator agent does not necessarily have to send it to the appropriate agent; it can send it to its neighboring agent. If this neighboring agent cannot fulfill the query, it would forward the query to the next agent, and so on. Upon receiving a response to the query, the original agent needs to update its knowledge base. In this way, when it submits the same type of query for processing the next time, it can direct the query to the appropriate agent.

Interface agents interact with the users, accept a user query, transform it into the proper language used within the system and send the transformed query to the appropriate mediator agent. When sending a user query to mediator agents, interface agents may submit a user profile as well, so that mediator agents can search for information that corresponds to the user's preferences. The function of a source agent is similar to that of wrappers as mentioned in the mediated approach.

3.2.1.2. Data management

While System Integration deals with the integration of different heterogeneous sources, Data Management is important for providing support for managing vast quantities of data representing each digital object to transform them into useful knowledge. The increasingly large amount of data that is being generated by applications in domains such as satellite imaging, high energy physics and computational genomics, has led to data volumes being measured in terabytes, and soon petabytes. The access patterns and types of uses of such data in scientific computing have generally differed from those in business computing. Whereas in business computing the emphasis appears to be on persistent data (such as customer records, product information, supplier details), in scientific computing the emphasis is on the ability to access data in large blocks, which are generally non-persistent. The ability to process and manage data involves a number of common operations, the extent of which depends on the application. Hence, data management generally involves:

- *Data pre-processing and formatting*: for translating raw data into a form that can be usefully analysed. Data processing may involve transforming a data set into a pre-defined range (for numeric data), and identifying (and sometimes filling in) missing data, for instance. The data processing stage is generally part of the data quality check, to ensure that subsequent analysis of the data will lead to meaningful results.

  Metadata is generally used in this context, for translating data from one form to another. Metadata can correspond to the structure of a data source, such as a database schema, which enables multiple data sources to be integrated. Alternatively, metadata may be summary data which identifies the principal features of the data being analysed, corresponding to some summary statistics. Generally, summary statistics have been generated for numeric data, however extensions of these approaches to

data that is symbolic is a useful current extension. This involves identifying syntactic or context based similarities between records within a database.

- *Data fusion*: for combining different types of data sources, to provide a unified data set, that could provide more useful insights into an experiment. Data fusion generally requires a pre-processing stage as a necessity, in order for data generated by multiple experiments to be efficiently integrated. An alternative to fusion is *Data splitting*, where a single data set is devided to facilitate processing of each sub-set in parallel.

- *Data storage*: involves the recording of data on various media, ranging from disks to tapes, which can differ in their capacity and "intelligence". Data storage can involve data migration between different storage media (based on a Hierarchical Storage Management (HSM) system), which vary based on access speed to storage capacity. Specialised applications, such as scientific visualisation, require specialised data storage to enable data to be shuffled between the application program and secondary (or even tertiary) storage at a faster rate, compared to other data processing applications. Data storage hardware and software also differ quite significantly, based on the particular domain requirements. Hence, hardware resources (and software support for them) can vary from RAID (Redundant Array of Independent Disks) drives, where support is provided for stripping data across multiple disks and/or parity-checks to ensure that lost data can either be reconstructed or migrated when a disk fails, to large scale data storage units such as High Performance Storage System (HPSS)[149] from IBM and products from FileTek[52] and AMPEX[7].

- *Data analysis*: can range from analysing trends in pre-recorded data for hypothesis testing, to checking for data quality and filling in missing data. Data analysis is an important aspect of data management, and has been successfully employed in various scientific applications. Analysis approaches can range from evolutionary computing approaches such as neural networks and genetic algorithms, rule based approaches based on predicate/propositional logic to Case Based Reasoning (CBR) systems, to statistical approaches such as regression. The data analysis approach generally requires a prior data preparation (pre-processing) stage.

- *Visualisation, navigation and steering*: is the emerging area within data management, that can range in complexity from output display on desktop machines to specialised visualisation and semi-immersive environments such as ImmersaDesk[83] and CAVE[25]. Visualisation tools such as IRIS Explorer[84] and Data Explorer[40] have been widely used in the scientific community, and provide a useful way to both generate new applications, and for visualising the results of these applications. The next stage (providing computational steering support) will enable scientists to interact with their simulation in real time, and dynamically *steer* the simulation towards a particular parameter space. Visualisation therefore becomes an enabler in creating and managing new types of scientific experiments, rather than as a passive means for viewing simulation output.

Data management is therefore a unified process that involves a number of stages, and it is important to view it as a whole. Each individual stage within the process has its own family of products and algorithms.

3.2.2. Agent technology

As discussed in *section 3.2.1.1.1.3*, the agent-based approach is one of the most promising solutions for System Integration in digital libraries. Agent technology has been used to address both System Integration and Data Management in the content of a digital library.

3.2.2.1. What is an agent ?

An agent can be perceived as a software entity which acts analogous to a human agent (decision maker). Consider for instance the role undertaken by a travel or estate agent. Their primary objective is to achieve a task and they both act on behalf of others; in the case of the estate agent, the agent acts on behalf of the actual owner of the property, whereas the travel agent acts on behalf of the hotels and flight companies. Acting on behalf of another entity is the first fundamental property of agency. A second fundamental characteristic of agents is that they both enjoy at least some degree of autonomy; for instance, estate agents can generally make viewing appointments for unoccupied properties without reference to the owners. A third important aspect of an agent's behavior is the degree of proactivity and reactivity present in their behavior. In this instance, an estate agent who simply places a "For Sale" sign outside a property for sale and waits for purchasers to come into his/her shop is behaving in a much

more reactive fashion, than an agent who proactively advertises the property in the local press. It should be noted however that reactivity and proactivity are not flip sides of the same coin. The same agent can display high amount of both proactivity and reactivity at different times[74]. Some properties that the agents may possess in various combinations include[108] the ability to be:

- *Autonomous*	is capable of acting without direct external intervention. Has some degree of control over its internal state and actions based on its own experiences.
- *Interactive*	communicates with the environment and other agents.
- *Adaptive*	capable of responding to other agents and/or its environment to some degree. More advanced forms of adaptation permit an agent to modify its behavior based on its experience.
- *Sociable*	interaction that is marked by friendliness or pleasant social relations, that is, where the agent is affable, companionable, or friendly.
- *Mobile*	able to transport itself from one environment to another.
- *Proxy*	may act on behalf of someone or something, that is, acting in the interest of, or as a representative of, for the benefit of some entity.
- *Proactive*	goal-oriented, purposeful. It does not simply react to the environment.
- *Intelligent*	state is formalised by knowledge (e.g. BDI model - Beliefs Desires Intentions) and interacts with other agents using symbolic language. An agent's *beliefs* correspond to information the agent has about the environment in which it is operating, *desires* correspond to the tasks allocated to it (its goals), and *intentions* represent desires that it has committed to achieving.
- *Rational*	able to choose an action based on internal goals and the knowledge that a particular action will bring it closer to its goals.
- *Unpredictable*	able to act in ways that are not fully predictable, even if all the initial conditions are known. It is capable of nondeterministic behavior.
- *Temporally continuous*	is a continuously running process.
- *Character*	believable personality and emotional state.
- *Transparent and*	must be transparent when required, yet must provide a log of its activities upon

accountable	demand.
- *Coordinative*	able to perform some activity in a shared environment with other agents. Activities are often coordinated via a plan, workflow, or some other process management mechanism.
- *Competitive*	able to coordinate with other agents except that the success of one agent implies the failure of others (the opposite of cooperative).
- *Robust*	able to deal with errors and incomplete data robustly.
- *Trustworthy*	adheres to Laws of Robotics and is truthful.

It is not necessary for an agent to adhere to all of the above properties at the same level. Consequently, different forms of agents exist such as software, coordinative, interactive, intelligent etc. which can be more easily thought as roles that an agent can play rather than the fundamental approach designed into an agent. Other forms of agents not named by the properties of an agent are facilitator, broker, management, wrapper etc. For instance, wrapper agents allow another agent(s) to connect to a non-agent software system/service uniquely identified by a software description. User agents can relay commands to the wrapper agent and have them invoked on the underlying services. The role of a wrapper agent provides a single generic way for agents to interact with non-agent software systems. It provides a bridge to legacy code and facilitates the reuse of code for an agent's process.

3.2.2.2. Mobile agents

While stationary agents exist as a single process on one host computer, mobile agents can pick up and move their code to a new host where they can resume executing. Historically, they are based on work carried out in the 80s on process migration and on distributed object computing[9][46][93][136]. The combination of the two areas, i.e. to migrate distributed objects was first reported in[93]. However, with the spread of the Java programming language, researchers became widely interested in object mobility. Java has been crucial for the development of mobile agents, as it has been designed as an architecture independent, network centric programming language, which provides many of the requirements to implement object mobility as a standard feature[70].

The rationale for mobility is the improved performance that can sometimes be achieved by moving the agent closer to the service(s) available on a host. However, if the volume of information exchanged with the remote site is large, issues of traffic and bandwidth must be considered. Also, the agent might be able to process the remote data more effectively than those services offered at the remote site. In either or both of these cases, relocating the agent to each of the various platforms could be a more efficient way of remote data processing. One disadvantage of such mobility is that the remote sites must provide an environment in which the mobile agent can reside and perform. This not only brings an additional processing burden to the remote site, but also raises three important issues: security, managing the load of visited agents, and unanticipated scalability problems. Nevertheless, mobility is an important property for many agent-based systems and necessary for a certain class of application.

The general structure of a mobile agent is illustrated in Figure 3.1. The core is based on the computational model and has significant impact on the other models since it defines how a mobile agent executes when it is in a *running* state. Both the security and life cycle models are structurally very close to the core. Security issues permeate every aspect of a mobile agent and therefore must be provided for at the most basic level. The life cycle model defines the valid states for an agent - represented in Figure 3.2. The outer layer contains the communication, navigation and agent model. The agent model defines the intelligent agent aspects of a mobile agent, such as learning. The communication model defines the ability of an agent to communicate with other entities, including other agents (static or mobile), services and users. Finally, the navigation model concerns the agent's mobility aspect, from the discovery and resolution of destination hosts to the manner in which a mobile agent is transported.

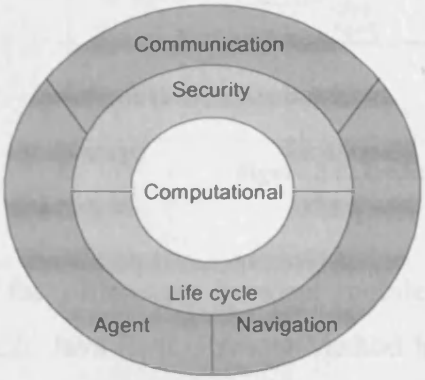


Figure 3.1. Structure of a mobile agent[74]

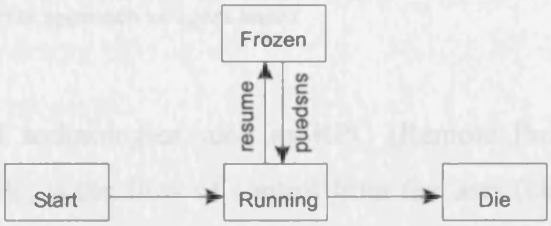


Figure 3.2. Life cycle of a mobile agent[74]

3.2.2.3. What makes mobile agents distinctive?

As the name suggest, mobile agents are programs that encapsulate data and code, which may be dispatched from a user computer and transported to a remote host for execution[29][77][163]. When large quantities of data are stored at distributed remote hosts, moving the computations to the data is a more realistic and feasible approach, compared to migrating data to the computations. Instead of gathering data distributed in remote sites at a centralised site, users can dispatch mobile agents to a destination site to perform information retrieval and filtering locally, and return to a user the result of analysis. Local messages are 1,000 to 100,000 times faster than remote messages[124].

Since the flow of control is not tied-up with the user by using mobile agents, the user does not need a permanent connection to the network until the results of analysis are generated. In consequence, the network traffic is reduced, the server load is minimised and the user connection costs (to an ISP - Internet Service Provider) are cut-down enormously[148]. Mobile agents are the best solution for networks with unreliable connections or narrow bandwidth since the information transmitted over the network is minimised.

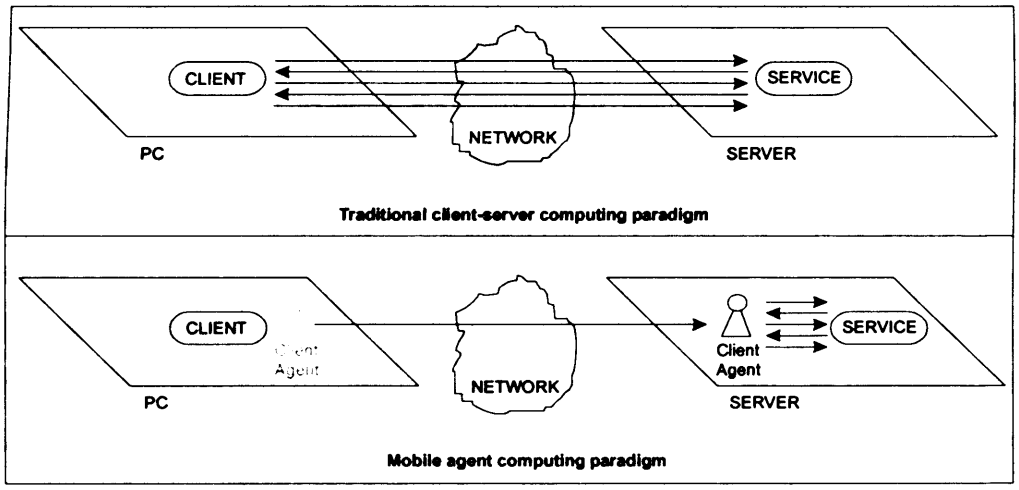


Figure 3.3. Traditional client-server approach vs agent-based

One of the differences between mobile agents and technologies such as RPC (Remote Procedure Call)/DCE, Java RMI (Remote Method Invocation) etc, is the flow of control from the user (client) to the server. In standard RPC/RMI approaches, although the user invokes a remote service, the control is always with the user. In mobile agents, control is not associated with a user, but moves as the agent

migrates. Comparisons of mobile agent approaches and RPC/RMI have shown dramatical differences in term of scalability and efficiency. A theoretical analysis of the trade-off between mobile agent migration and the remote procedure call paradigms can be found in [143]. A performance evaluation of an agent-based home banking system in contrast with the corresponding RPC-based system can be found in [148]. Figure 3.3 represents the traditional client-server approach (RPC paradigm) versus the agent-based one. While in the first approach the user is bound directly to the server; in the second, the user is free to engage with other tasks once it has dispatched its mobile agent to the server. When the mobile agent accomplishes its task on the server-side, it migrates back to the user or sends directly the results.

Mobile agent transactions are robust and flexible. Once a user has created an agent, it can run autonomously and asynchronously, without intervention from the user. Mobile agents provide a reliable transportation between a client and a server without necessitating a reliable underlying communication medium. They can also react autonomously to changes in their environment, and are therefore more flexible in their operation. The capability of communicating by exchanging synchronous/asynchronous multicast or broadcast messages makes mobile agents more attractive to developers. Mobile agents have also been introduced to Peer-to-peer (P2P) systems to perform operations at peers' sites. P2P networks are emerging as a new distributed computing paradigm for their potential to harness the computing power of the hosts composing the network and make their under-utilised resources available to others and has attracted enormous attention from the emerge of file sharing systems (such as Napster[175], Morpheus[174], eDonkey[173], eMule[172] etc.)

3.2.2.4. Multi-Agent systems

Research into systems composed of multiple agents was initially carried out under the banner of Distributed Artificial Intelligence (DAI), and has historically been divided into two main camps[15]: Distributed Problem Solving (DPS) and Multi-Agent Systems (MAS). More recently, the term "multi-agent systems" has come to have a more general meaning, and is now used to refer to all types of systems composed of multiple (semi-) autonomous components.

DPS considers how a particular problem can be solved by a number of entities, which cooperate in dividing and sharing knowledge about the problem and its evolving solutions. In a pure DPS system, all

interaction strategies are incorporated as an integral part of the system. In contrast, research in MAS is concerned with the behavior of a collection of possibly pre-existing autonomous agents aiming at solving a given problem. A MAS can be defined as a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver[47] (agents) which are autonomous and may be heterogeneous in nature, i.e. hosted on different kinds of platforms. Therefore, the characteristics of a MAS are: (a) each agent has incomplete information or capabilities for solving the problem i.e. each agent has a limited viewpoint, (b) there is no global system control, (c) data is decentralised, and (d) computation is asynchronous.

The increasing interest in MAS research includes[74][91] their ability to:

- solve problems that may be too large for a centralised single agent to do due to resource limitations or the sheer risk of having one centralised system.

- allow for the interconnecting and interoperation of multiple existing legacy systems e.g. digital libraries, expert systems, decision support systems, GIS etc.

- provide solutions to inherently distributed problems, e.g. air traffic control or solutions where the data, control or expertise is distributed e.g. in health care provision.

- provide solutions which draw from distributed information sources, when software and hardware resources are distributed.

- enhance:
  (a) speed: since communication is kept minimal due to the use of mobile agents,
  (b) reliability: capability to recover from the failure of individual components, with graceful degradation in performance,
  (c) extensibility: capability to alter the number of agents applied to a problem.

- offer conceptual clarity and simplicity of design.

In a mutli-agent system, agents need to communicate among themselves, cooperate, coordinate their activities and negotiate once they find themselves in conflict. In heterogeneous distributed systems it is impossible to create, move and run arbitrary software objects on remote machines. This is the purpose of having agent environments i.e. an agent platform that stands as the home of agents where they can be created, transferred, execute, communicate and terminated. Security considerations also demand protection mechanisms to defend the operating system against possible malicious actions resulting from the actions of imported agents, and are usually provided by the software of the agent platform engaged.

3.2.2.5. Usage of agents

In [91] there is a review of agent-based systems categorised in four main sections according to the nature of application realised, these include:

- *Industrial applications*: Industrial applications of agent technology were among the first to be developed, and today, agents are being applied in a wide range of industrial systems, such as manufacturing, process control, air traffic control, telecommunications and transportation systems.

- *Commercial applications*: While industrial applications tend to be highly-complex, bespoke systems which operate in comparatively small niche areas, commercial applications, especially those concerned with information management (the gathering and filtering of information), tend to be oriented much more towards the mass market. Other applications in this area include electronic commerce and business process management.

- *Entertainment applications*: Agents have an obvious role in computer games, interactive theatre, and related virtual reality applications. Such systems tend to be full of semi-autonomous animated characters, which can naturally be implemented as agents.

- *Medical applications*: Medical informatics is an important and major growth area in computer science. Agent-based applications in this area have been developed for patient monitoring and distributed health care.

3.2.3. Agent communication language

It is essential that agents used to access heterogeneous remote data archives communicate and co-operate with each other in order to provide service and satisfy user requests within the predefined constraints. A simple way to do this is to define an interaction protocol for communication in the particular problem. The best way to represent such a protocol and to define a standard message format with meaningful structure and semantics has become a key issue.

Most agent communication languages such as KQML (Knowledge Query Meta Language)[96] and FIPA ACL (Agent Communication Language)[54] have been designed to minimise the size of the message and to function more as a data-passing protocol. Little emphasis has been placed on the flexibility or the transparency of the semantics of the message. Many other agent communication languages use the basic KQML/FIPA ACL style, but replace or extend the sets for special purposes, such as contract negotiation, offers, and bids.

XML (eXtensible Markup Language) is becoming the standard for data interchange on the internet, and enables Web Services that are not meant for direct use by humans, but rather to be used by other software. Its flexibility and ability to clearly represent and identify (or describe) data makes XML ideal for transferring data between agents. The KQML and FIPA ACL agent communication languages, the most well known, have been converted to simple XML form. Several XML-based schemas are being designed, such as Resource Description Framework (RDF)[125], XML-Data[165] and Document Content Definition (DCD)[45]. Most existing XML schemas focus on strong data formats. For instance, RDF focuses on how to represent semantic networks; XML-Data considers basic data types such as Integer, Long and Date; DCD is a simplification of RDF that takes account of the data types of XML-Data.

3.2.3.1. XML

XML[164] has been in development since the 1960s through its parent called SGML (Standard Generalised Mark-up Language), which was set as an international standard in 1986 as the basis for structured document publishing. In the mid-1990s, an SGML application called HTML (Hyper Text Mark-up Language) emerged as the main publishing method for large-scale electronic documents on the

WWW (World Wide Web) . In 1996 a working group in the WWW consortium started developing XML as a streamlined version of SGML. XML was designed for transmission of structured data over the web, retaining the powerful structured concept of SGML but removing portions that are very complex and have limited application.

XML is both simple and powerful. It is designed to improve the functionality of the web by providing more flexible and adaptable information identification[171]. It is called "extensible" because it is not a fixed format like HTML - a single, predefined markup language. Instead, XML is actually a meta-language i.e. a language for describing other languages, which enables a new generation of web services that are not meant for humans to use directly, but rather to be used by other services. XML is not just for web pages, it can be used to store any kind of structured information, and to enclose or encapsulate information in order to pass it between different computing systems which would otherwise be unable to communicate.

In general, an XML document consists of the following three parts:

- *Structure*: defines the document type and the organisation of its elements. A set of rules exists in order to enforce what kind of elements it contains, in what order they occur, and what additional attributes of elements are allowed.

- *Presentation*: concerns the way information is presented on a web-page i.e. whether a block of text is in bold or italic, which fonts to use etc.

- *Data content*: regards the informational data contained in a document.

3.2.4. Compatibility in Multi Agent Systems

The highly interactive nature of multi-agent systems points to the need for consensus on agent interfaces in order to support interoperability between different agent systems. The completion and adoption of such a standard is a prerequisite to the commercialisation and successful exploitation of intelligent agent systems.

The sub-section below briefly discusses and compares the most important efforts that define interoperability between agents on different types of platforms, with emphasis on FIPA efforts.

3.2.4.1. Different approaches of standardisation

Currently, there are three important agent standardisation efforts that define interoperability between agents on different types of platforms[120][121]. KQML community, OMG's MASIF (Mobile Agent System Interoperability Facility) and FIPA.

KQML[96] was one of the first initiatives to specify how to support the social interaction characteristic of agents using a protocol based on speech acts. KQML was developed at UMBC by Tim Finin et al[53] and has spread throughout the academic community. KQML however is not a true de facto standard in the sense that there is no consensus on a single specification or set of specifications that it has been ratified by common agreement within an organisation or forum of some standing in the community. As a result, variations of KQML exist such as KQML classic, KQML '93 and KML-Lite, leading to different agent systems that speak different dialects and that are not able to interoperate fully.

MASIF[101] differs from both KQML and FIPA in that it regards the defining characteristic for an agent as its mobility from one location to another. MASIF does not support the standardisation of communication between agents on different agent platforms. Furthermore, MASIF restricts the interoperability of agents to those developed on CORBA platforms whereas the focus of FIPA is to directly support the interoperability of agents deployed on agent frameworks which can support different message transport protocols. OMG is exploring how to support other characteristics of software agent than mobile agents and it issued a "Request For Information" (RFI) on agents in 1999[110]. FIPA has supplied its specifications as input to this request. This is still work in progress at this time.

FIPA[54] was formally established as an international non-profit association of companies and organizations which agree to share efforts to produce specification for generic agent technologies with the following features:

- *Timely* (that is, the time to reach consensus and to complete the standards should not be long, and it should not act as a brake on progress rather than an enabler, before industries make commitments)

- *Internationally agreed*

- *Usable across a large number of applications*

- *Yielding high level of interoperability across applications*

The standardisation work of FIPA is intended to allow an easy interoperability between agent systems, because FIPA beyond the agent communication language specifies also the key agents necessary for the management of an agent system, the ontology necessary for the interaction between systems, and it defines also the transport level of the protocols (unlike KQML). The use of a common communication language, such as KQML, is not enough to easily support interoperability between different agent systems. The core mission of the FIPA standards consortium is to facilitate the interworking of agents and agent systems across multiple vendors' platforms. This is expressed more formally in FIPA's official mission statement.

"The promotion of technologies and interoperability specifications that facilitate the end-to-end interworking of intelligent agent systems in modern commercial and industrial settings"[54]

FIPA initially announced "FIPA 97" specifications, later on "FIPA 98" and nowadays "FIPA 2000". A FIPA agent platform is defined as software that implements the set of FIPA specifications. To be considered FIPA-compliant, an agent platform implementation must at least implement the "Agent Management" and "Agent Communication Language" specifications, which should conform to the latest *experimental* and/or *standard* status specifications i.e. FIPA 2000.

In contrast to MASIF, both KQML and FIPA emphasise agency and social interaction between multiple agents as the defining properties for software agents. They both define interaction in terms of an Agent Communication Language (ACL) whereas MASIF defines interaction in terms of Remote Procedure

Calls (RPC) or Remote Method Invocation (RMI). In contrast to the traditional RPC-based paradigm of MASIF, the ACL as defined provides an attempt at a universal message-oriented communication language. The FIPA ACL describes a standard way to package messages, in such a way that it is clear to other compliant agents what the purpose of the communication is. Although there are several hundred verbs in English, which correspond to performatives, the ACL defines what is considered to be the minimal set for agent communication (FIPA ACL consists of 20 or so performatives).

One important trend in the FIPA standard is away from the specification of single external interfaces to multiple external interfaces. For instance, FIPA in early versions of its specification defined a single so called base-line "transport protocol" - OMG's Inter-ORB Protocol (IIOP). This in essence means that FIPA agent platforms can also run on top of CORBA. There was a growing realisation that one transport protocol was not suitable for all domains, for instance, an interface has been defined to a WAP (Wireless Application Protocol) transport and more may follow. Similarly, FIPA97 specified a single ASCII string encoding for the ACL message but FIPA now specified multiple encodings such as Unicode and other text language encodings such as XML and binary (bit-efficient) encodings.

To conclude, standards need to be developed at the right time and FIPA seems to adapt towards to the technological improvements. FIPA specifications are not arbitrarily set, they have a life-cycle and in order for a specification to become standard two years of experimental tests must pass. Since FIPA was established the membership of companies and organisations has been increasing. In 1996 FIPA consisted of 25 companies (5 of which were Universities), while in the early 2001 this increased to 63, and nowadays to 71.

The increase of the members incorporated into FIPA, the presence of companies such as IBM, NASA (National Aeronautics and Space Administration), Intel, Philips etc., and the utilisation in large-scale projects (such as AgentCities[1], Facts[51], Cameleon[24]) based on FIPA's specifications; are factors that are likely to contribute to making FIPA specifications a universally accepted standard.

3.2.4.2. Overview of the FIPA specifications

Since January 2000, FIPA has adopted a new procedure for classifying, organising and releasing specifications to ensure coherence, completeness and consistency of its work as well as its relevance to industrial and commercial interests[39]. It is important to note that FIPA specifications do not attempt to describe how developers should implement their agent-based systems, nor do they attempt to specify the internal architecture of agents. Instead, they provide the interfaces though which agents can communicate. Each specification is given a subject association that describes the general area in which it belongs in the FIPA specification structure, depicted in Figure 3.4. FIPA specifications are divided into five categories: Applications, Abstract architecture, Agent Communication, Agent Management and Agent Message Transport, which are briefly described below. Each area of specifications has one or more specification documents assigned to it, which can be downloaded from FIPA's web-site[54]. In the remainder of this section, FIPA specification documents are referenced by their unique ID number, as assigned by FIPA, enclosed in curly brackets. A complete list of the FIPA specification documents referenced within this dissertation can be found in *Appendix A5*.

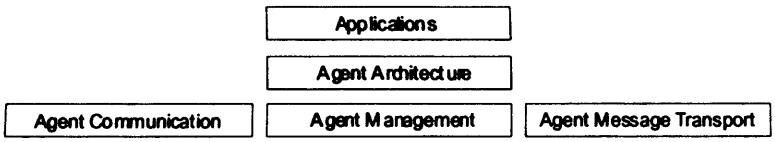


Figure 3.4. FIPA Specifications breakdown[54]

Abstract Architecture: The purpose of the FIPA Abstract Architecture {FIPA00001} is to foster interoperability and reusability. To achieve this, it is necessary to identify the elements of the architecture that must be codified. Specifically, if two or more systems use different technologies to achieve some functional purpose, then it is necessary to identify the common characteristics of the various approaches. This leads to the identification of architectural abstractions i.e. abstract designs that can be formally related to every valid implementation. By describing systems abstractly, the relationships between fundamental elements of agent systems can be explored. By describing the relationships between these elements, it becomes clearer how agent systems can be created so that they are interoperable. From this set of architectural elements and relations, a broad set of possible concrete architectures can be derived, which will interoperate due to the fact that they share a common abstract design. Furthermore, because an abstract architecture permits the creation of multiple concrete

realisations, it must provide mechanisms to permit them to interoperate. This includes providing transformations for both transport protocols and message encodings, as well as integrating these elements with the basic elements of the environment. The FIPA Abstract Architecture makes a distinction between those elements which can easily be defined in an abstract manner, such as agent message transport, FIPA ACL, directory services and content languages, and those elements that cannot, such as agent management and agent mobility. These are considered difficult to represent abstractly since they occur too close to the concrete realisation (implementation) of an agent system and very little commonality can be derived from analysing them. Yet, these issues will have to be addressed by developers and the abstract architecture will provide a number of instantiation guidelines in the future for specific groupings of implementation technologies. The first concrete realisation of the FIPA Abstract Architecture is the JAS (Java Agent Services)[87] project that is being developed as part of the Java Community Process, which is still in progress at this time.

Agent Message Transport: The FIPA Agent Message Transport specification {FIPA00067} deals with the delivery and representation of messages across different network transport protocols, including wireline and wireless environments. At the message transport level, a message consists of a message envelope and a message body. The envelope contains specific transport requirements and information that is used by the Message Transport Service (MTS) on each agent platform to route and handle messages. The message body is the real payload and is usually expressed in FIPA ACL but is opaque to the MTS since it may be compressed or encoded. The Agent Message Transport reference model depicted on Figure 3.5, provides facilities for:

- General support for an MTS within an agent platform {FIPA00067}

- Guidelines for using specific Message Transport Protocols (MTPs), such as IIOP {FIPA00075}, HTTP (Hyper Text Transport Protocol) {FIPA00084} and WAP {FIPA00076}

- Message envelope representations that are suitable for each MTP, such as an XML encoding for HTTP {FIPA00085} and a bit-efficient encoding for WAP {FIPA00088}

- FIPA ACL representations, such as a string encoding {FIPA00070}, an XML encoding {FIPA00071} and a bit-efficient encoding {FIPA00069}

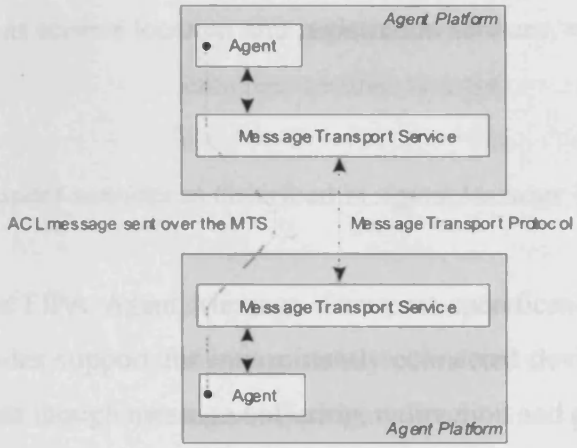


Figure 3.5. Agent Message Transport Reference model {FIPA00067}

The MTS on each agent platform can support any number of message transport protocols and will normally translate between a FIPA-supported MTP (Message Transport Protocol) that is used for interoperable communication between heterogeneous agent platforms such as XML over HTTP, and an MTP that is used internally to the agent platform such as Java objects over the Java Messaging Service. Consequently, the components of the MTS are designed to be modular and extensible to handle different message transport protocols, message envelope and FIPA ACL representations in the future.

Agent Management: The FIPA Agent Management specification {FIPA00023} provides the framework within which FIPA agents exist and operate. It establishes the logical reference model for the creation, registration, location, communication, migration and termination of agents. The entities contained in the agent management reference model depicted in Figure 3.6, are logical capability sets i.e. services and do not imply any physical configuration. Additionally, the implementation details of agent platforms and agents are the design choices of the individual agent system developers. The reference model describes the primitives and ontologies necessary to support the following services in an agent platform:

- *White pages*, such as agent location, naming and control access services, which are provided by the Agent Management System (AMS). Agent names are represented by a flexible and extensible

structure called an agent identifier, which can support social names, transport addresses, name resolution services, amongst other things.

- *Yellow pages*, such as service location and registration services, which are provided by the Directory Facilitator (DF).

- *Agent message transport services* as described in *Agent Message Transport* above.

In conjunction with the FIPA Agent Message Transport specifications, the FIPA Agent Management specification also provides support for intermittently connected devices, such as laptop computers and personal digital assistants though message buffering, redirection and proxying.

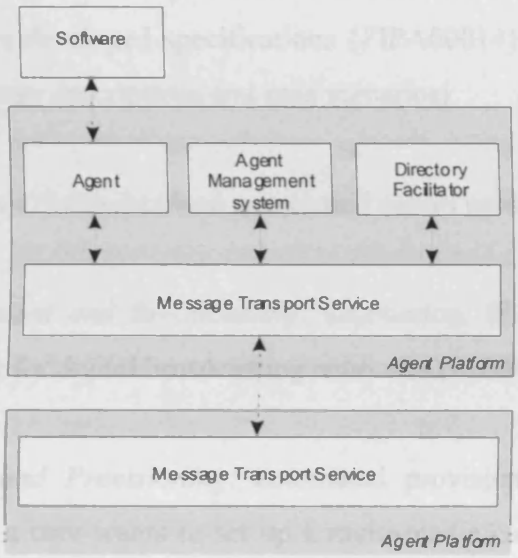


Figure 3.6. Agent Management Reference model {FIPA00023}

Agent Communication: Communication between agents in FIPA is based on a model of semantically grounded communication {FIPA00061} i.e. communication that is pre-defined, semantically rich and well understood by agents. The basis of communication between FIPA agents is through the use of communicative acts, that are based on speech act theory[10][133]. Communicative acts are verbs denoting a speech act which enables a receiving agent to understand in which context to interpret the contents of the enclosed message.

FIPA specifies a number of communicative acts or "performatives" {FIPA00037}, such as *request*, *inform* and *refuse* in a well-defined manner that is independent from the overall content of the message. The message that is supplied with a communicative act is itself wrapped in a well-specified envelope, called an Agent Communication Language (ACL). ACL provides mechanisms for adding context to the message content, the sender and receiver, the ontology and interaction protocol of the message. FIPA ACL {FIPA00061} was originally based upon ARCOL[127] with a number of revisions from KQML[96]. The actual content of a message is expressed in a content language, such as the FIPA semantic language {FIPA00008}, a constraint choice language {FIPA00009}, KIF (Knowledge Interchange Format) {FIPA00010}, RDF {FIPA00011} or XML. Finally, the set of FIPA interaction protocols {FIPA00026-FIPA00036} describe entire conversations between agents for the purpose of achieving some interaction or effect, such as auctioning, issuing a call for proposal, negotiating brokering services and the registration/deregistration of subscriptions.

Agent Applications: FIPA has developed specifications {FIPA00014} of four agent-based applications that contain service and ontology descriptions and case scenarios:

- *Personal Travel Assistance*: individualised, automated access to travel services {FIPA00080}

- *Audio-Visual Entertainment and Broadcasting*: negotiating, filtering and retrieving audio-visual information, in particular for digital broadcasting networks {FIPA00081}

- *Network Management and Provisioning*: automated provisioning of dynamic Virtual Private Network services where a user wants to set up a multi-media connection with several other users {FIPA00082}

- *Personal Assistant*: management of a user's personal meeting schedule, in particular in determining the time and place arrangements for meetings with several participants {FIPA00083}

3.2.5. Management of agents within a Multi Agent System

Agent management is the key element in every multi-agent system. Essentially, management is a process in which agents engage in order to ensure a community of individual agents act in a coherent and

harmonious manner. Perhaps, the easiest way of ensuring coherent behavior and resolving conflicts seems to consist of providing the group with a management agent, which has a wider perspective of the system. There are many reasons why multiple agents need to be managed. No agent possesses a global view of the entire agency to which it belongs, as this is simply not feasible in any community of reasonable complexity. Consequently, agents only have local views, goals and knowledge, which may interfere with and support other agents' actions. Management is vital to prevent *chaos* during conflicts and failures. Agents possess different capabilities and expertise. Therefore, agents need to be managed in order to cooperate and serve their goals. Agent actions are frequently interdependent and hence an agent may need to wait on another agent to complete its task before executing its own, and such interdependent activities need to be managed as well.

Successful management is based on the management agent's information about the system, the agent's desires and the organisation techniques applied according to this knowledge. System state information corresponds to the status of each server, the availability of resources, the distribution of agents (load balancing) on the network, any conflicts/failures or updates taking place on the system; where agents' desires are the users' requests.

Load balancing (LB) is one of the most important techniques that can be applied to support the management of agents within a MAS, because apart from the even distribution of agent tasks among the servers, the management agents' information on LB may also be reused by other techniques that can extend the scalability of a MAS. For instance, optimizing mobile agents' migration is feasible due to the management agents' global view of the system, intelligent techniques can be used for breaking an agent's task into smaller sub-tasks and assigning them to multiple agents for parallel execution. Caching techniques are possible to be applied based on statistics generated by the management agents from comparing agent's desires for identification of request similarities. Moreover, the agents' details - gathered by the LB management agents - along with the system information provides the foundations for an efficient monitoring mechanism for observing and improving the performance and reliability of large scale distributed systems.

3.2.5.1. Load balancing of mobile agents

Generally, load balancing aims to improve the utilisation and performance of tasks on available servers, whilst observing particular constraints on task execution order. Assuming agents have a set of tasks to execute, it is necessary to identify how these tasks may be distributed across available servers. Hence, although there may be an even distribution of agents among the servers, the load on the servers may not be balanced due to the different amounts of work undertaken by each agent.

Load balancing can be either static or dynamic[70] according to the multi-agent system in which it is being considered. In static load balancing tasks cannot be migrated elsewhere once they have been launched on a specified server. In dynamic load balancing a task may migrate to another server, utilising the agent's mobility.

There are two basic approaches to distribute tasks among servers: the state-based and the model-based approach. In the state-based approach, information about the system state is used to determine where to start a task. The quality of this decision depends on the amount of the state data available. Gathering the data is expensive, but leads to a more accurate decision. In the model-based approach, load balancing depends on a model which predicts the system state and which may be inaccurate. Model-based approaches to load balancing are much rarer, as they involve the derivation of an initial model, and the need to adapt the model over time. No work exists on integrating the state and model-based approaches for load balancing i.e. to construct and adapt the model with minimal state information. Further discussion on state and model-based approaches for load balancing may be found in *section 3.3.3*.

3.3. Related work

The theoretical part of this dissertation has been covered by the previous sections. The remainder of this chapter is divided in three main *sub-sections 3.3.1, 3.3.2* and *3.3.3*, where each one provides a review of the most important efforts on the field of digital libraries, interoperability of multi-agent systems and load balancing from which the research on SARA architecture has been influenced.

3.3.1. Digital Libraries

Digital libraries has become a popular topic for many research groups since the early and mid 1990's with the high level of attention and funding being given to digital libraries[66]. The Digital Libraries Initiative (DLI) was announced in late 1993, whose focus is to dramatically advance the means to collect, store, and organise information in digital forms, and make it available for searching, retrieval, and processing via communication networks in user-friendly ways. With the selection and funding of the six DLI projects, interest and activities related to digital libraries accelerated rapidly[139][140].

The University of Michigan Digital Library (UMDL)[14][38] utilises highly specialised information agents to perform information retrieval across heterogeneous sources. Each agent has two properties: autonomy and negotiation. The UMDL architecture consists of a cooperating set of three types of software agents: user interface agents, mediation agents, and collection agents. User interface agents conduct interviews with users to establish their needs such as what they need to know, and the breadth and depth of the information they require. The interface agent enables the user to specify areas of interest so that the system can notify the user of items of potential relevance. Mediation agents coordinate searches of many distinct but networked collections (wrapped by collection agents) by taking orders from the interface agents.

The UC Berkeley Digital Library Project[159] is aimed to develop the tools and technologies to support highly improved models of the "scholarly information life cycle", to facilitate the move from the current centralised, discrete publishing model, to a distributed, continuous, and self-publishing model. The Alexandria Digital Library (ADL) is focused on providing broad access to distributed collections of spatially-indexed information[8]. ADL architecture consists of four components: collections, catalog, interfaces and ingest facilities[67]. The Carnegie Mellon informedia digital video library project has focused on automated video and audio indexing, navigation, visualisation, search and retrieval and embedded them in a system for use in education, information and entertainment environments[30][31]. The University of Illinois at Urbana-Champaign (UIUC) digital library project mainly used web technology to effectively search technical documents on the Internet. An experimental test-bed with tens of thousands of full-text journal articles from physics, engineering, and computer science, and making these articles available over the World Wide Web, has been built for indexing of the contents of text

documents to enable federated search across multiple sources, testing this on millions of documents for semantic federation[16].

The Stanford Digital Library Project is aimed at resolving the issues of heterogeneity of information and services[115]. Based on CORBA technology, the Information Bus is the core system of the project that provides uniform access to heterogeneous information sources and services.

Rutgers University DigiTerra[106] is a space and land-based digital system, which consist of multiple layers of processors. The integration and interoperability layer is concerned with the collection and assimilation of a vast array of environmental data. XML have been chosen as a common language to be used by the mediators and wrappers to represent queries and responses. The ontology layer enables users with diverse backgrounds to query across multiple domains. The data warehousing/data-mining layer provides fast and efficient access to the integrated data, efficient data analysis, and historical, temporal and chronological views. The concept indexing and content-based retrieval layer provides efficient retrieval by suitably organising the multimedia data based on the concepts associated with the objects. The universal access layer provides methodologies to cater to diverse users' characteristics, preferences, and capabilities, as described earlier.

Virtual Community Library (VCL)[123] is a decentralised collection of interacting self-interested agents where an agent represents the knowledge and interests of an individual user. Each agent is perceived as a personal digital library serving one individual user. Within the VCL, the agents support the individuals' information acquisition and dissemination tasks information by querying other agents and interpreting the results according to the querying agent's knowledge, maintaining subscriptions and publication commitments according to the users' interests, providing speculative recommendations based on a framework for e.g. social of collaborative filtering.

Marchionini[99] has characterised DL research and development as falling into four categories: content, services, technology, and culture. Research issues related to content includes the integration of multimedia objects; data acquisition, including analog to digital conversion; metadata extraction and standardisation; indexing, storage and retrieval; workflow processes and management; and collection

preservation and maintenance. Service research issues are strongly dependent on user interfaces and include search, filtering and browsing; reference and question answering; and instruction. Technology research efforts are mainly related to high-speed networking, security and billing, and interoperability across many DLs. The culture issues include intellectual property; insuring data quality, privacy, and equity; and organisational interfaces for various communities of practice. In addition to these research and development challenges, meta issues related to managing and evaluating DLs and their impact on people and organisations are also active areas of study.

3.3.2. Efforts on the interoperability of MAS

FIPA specifications have been adopted by a wide range of companies, organisations and universities for the realisation of interoperable agent-based applications[56], and eleven major publicly available implementations of agent platforms[57] which conform to the FIPA Specifications have been developed. Previous work on an attempt to provide automated FIPA-interoperability in a legacy system has not been reported yet. The approach proposed in this thesis to conforming a legacy multi-agent system to a FIPA-compliant one which requires a developer to have no or limited knowledge of FIPA specifications is the first attempt towards this direction.

However, researchers have focused on interoperability aspects, such as the security and specially the mobility between heterogeneous agent-based systems, that have not yet been *successfully* addressed by FIPA. Even though there is currently debate as to whether a generic or default level of agent security ought to be specified, [119] defines the requirements and design issues for adding security to FIPA agent systems and proposes a secure agent platform model based on agent authentication using simple public key infrastructure and a private channel for transferring messages between agents when required. A new approach to agent mobility is presented in [18], called generative migration, where agents can migrate between non-identical platforms and need not be written in the same language. The key idea of generative migration is not to move the agent itself but to base migration on an implementation-independent description of the agent (called blueprint) that describe its compositional structure, functionality and state. A service called Agent Factory on each platform is capable of regenerating platform-depended agents based on receiving blueprints. Other approaches[97][104] separate the platform-independent part of an agent from the platform-specific part. Specifically [97] provides an API

called "Guest" for constructing agents. A Guest agent has two facets, the first one is specific to the platform (interface) expected to carry the agent and the other one is independent of any platform. As a result when a Guest agent moves from one platform to another one, it has only to change dynamically its platform-specific facet while maintaining its internal status. Currently the Guest API provides interfaces for the Voyager, Aglets, Grasshopper, Concordia, CorbaHost and JADE (Java Agent Development Framework) agent platforms, as well as for CORBA-implemented systems.

It is important to note that in 2000, the FIPA-NET[59] initiative - originating from the Intelligent and Interactive System group at Imperial College as part of its CASBA (Common Agent Service Brokering Architecture) project - was the first attempt to create a test-bed of multiple inter-linked FIPA agent platforms. It provided an operational FIPA multi-agent system together with information and tools to support interoperability between FIPA multi-agent systems distributed across the internet and offered a portal to link to services in other FIPA systems. Although FIPA-NET has been discontinued, it contributed to the AgentCities[1] initiative by providing part of the multi-agent infrastructure. The aim of AgentCities was to build a publicly accessible, continually available network of FIPA platforms. Each platform supports services modeled for a single real world city or place. Services deployed in the test-bed have been initially centered on information and transaction services for real world objects such as bars, restaurants, hotels, travel infrastructure, theaters etc. Agent-based applications can access these services worldwide using Federated Directory services (FDs) and FIPA communication services. The set of services deployed in the network can be used as building blocks to construct new agent services. Complex compound services such as planning a weekend away (organising flights and opera tickets, selecting restaurants, locating and booking a hotel and proposing an art exhibition to visit) has also been undertaken. The initial network of AgentCities platforms was deployed in October 2001 and up-to-date it engages organisations from more than 20 countries involved in a significant number of different projects.

3.3.3. Load balancing

As has been mentioned in *section 3.2.5.1* of this chapter, load balancing can be either static or dynamic where the distribution of tasks among the servers may be based on a state or a model. The following sub-sections discuss different approaches to load balancing according to the needs and properties of the system on which it is applied.

3.3.3.1. Static state-based load balancing

In state-based load balancing, a common approach for managing system state and load is the *market* mechanism to value resources and achieve an efficient match of supply and demand for resources. Some systems use only a price, and match offers and bids, while others employ more sophisticated auction protocols[128][94], such as *vickrey auction, sealed-bid double auction, repeated clearing-house double auction* etc. In these approaches consumers bid for resources according to the auction mechanism being employed. An advantage of auctions as a market mechanism is that they allow one to determine an unknown resource value in a group of agents. However, this agreement comes at a price; the communication needed to determine it.

Spawn[157] was the first system to employ market-based static load-balancing strategies. The auction protocols employed by Spawn are *sealed-bid* and *second-price*; these protocols combined together define a *vickrey* auction. *Sealed* means that bidding agents cannot access information about other agents' bids, and *second-price* indicates that the amount paid by the winning agent is equal to the second-highest bid placed. Buyers are represented by users (agents) who wish to purchase time in order to perform some computation and sellers are represented by workstations/servers who wish to sell unused, otherwise wasted processing time of their resources. Hence, a seller executes an auction process to manage the sale of its workstation processing resources, and a buyer executes a task that bids for time on nearby auctions. Apart from the fact that each workstation can only execute a single application task per time slice, Spawn suffers from other disadvantages as well. It does not provide tasks with robust recovery in the event of failure i.e. user computations can be aborted due to server failure or insufficient funding, and no attempt has been made to protect the Spawn economy from malicious users intent upon forging currency or deliberately cheating agents.

Other systems like Dynasty [11] and OCEAN (Open Computation Exchange & Auctioning/Arbitration Network)[109] avoid the communication overhead of auctions and use a pricing mechanism without any negotiation. For instance, Dynasty employs a hierarchical brokering architecture where the allocation of resources is based on dynamic pricing based on rent for utilising computing resources (which periodically varies), brokerage expenditures for getting assigned to a target host and fees for migration and data transport services. The local cluster broker determines several statistics like load indices, and

passes them up the hierarchy of brokers. Also, global knowledge is passed down. The brokers evaluate the qualification of their sub-brokers in order to allocate the tasks efficiently.

In contrast to market-based approaches, the MPI (Message Passing Interface)[114] and PVM (Parallel Virtual Machine)[68] message passing libraries can be used for static load-balancing. MPI's main motivation is portability of software for massively parallel processors, where PVM allows a user to view a network of heterogeneous hosts as a single large parallel computer. Therefore, while MPI cares for portability of software from one platform to another, PVM provides the infrastructure to make different, heterogeneous platforms transparently work together. PVM can adapt to variations in the utilisation of hosts and to re-configurations of the network, but the programmer is responsible for a good distribution of the parallel software among the hosts based on dynamic effects like background loads.

Finally, in CORBA[113] static load balancing can be achieved with a load balancer service which recommends services randomly to available hosts or by utilising runtime information, such as the amount of idle CPU available to choose the least loaded hosts.

3.3.3.2. Dynamic state-based load balancing

Keren and Barak[95] prove that dynamic load balancing achieves an improvement of 30-40% over a static placement scheme. They have focused on the migration of agents from over-loaded to less loaded hosts in order to distribute the load evenly among the hosts, and on the migration of intensively communication agents to common hosts in order to benefit from shared memory to minimise the communication overhead. Their scheme includes algorithms for on-line measurement of the resource utilisation, dissemination of load information among the hosts, and decision on migration of agents based on the collected information.

In the ACWN[166] dynamic load balancing strategy, tasks are divided into smaller sub-tasks which are then migrated to the least loaded neighbor host. Then each task is executed on the host which has been migrated, unless the load of a neighbor host is less than the former one. In that case the task is forwarded to the latter host. Thus a newly generated sub-task travels along the steepest load gradient to a local minimum. Each host is required to maintain its local load information, and adjacent hosts are needed to

exchange their load information periodically. Similar, to the ACWN's neighboring technique, in the LADE (Local Averaging-algorithm Dimension Exchange)[166] strategy a host's workload is balanced with respect to one of its neighbor hosts whereas in the LADF (Local Averaging-algorithm DiFfusion)[166] strategy a host manages to balance its workload with its neighbor hosts. Similar strategies for load balancing based on neighboring hosts can be found in [37].

In [19] a system is presented, which provides a market-based dynamic load-balance strategy for controlling the activities of mobile agents, using electronic cash, a banking system, and a set of resource managers. The agents carry with them a finite amount of currency, which they pay to resource managers for the resources they use. They can dynamically trade off space and time once they have seen the relative cost of the necessary resources, according to their own encoded priorities. The development of a distributed banking system is used to manage currency using on-line protocols that allow agents to talk to their banks while executing a transaction. The resource managers on the other hand, have the freedom to choose how to price their resources by using fixed (on initialisation) or dynamic pricing strategies (*sealed-bid second-price* auction) to allow them to adapt to the supply and demand in the system.

Other systems like MATS[69][22] and Traveler[163][167], use specialised agents to gather information about the system state. MATS (Mobile Agent Team System) proposes an approach to dynamic distributed parallel processing using a mobile agent-based infrastructure. It distinguishes three roles of agents, namely a Hive, Scout and Queen analogous to a bee colony. A Hive is responsible for managing user interaction and determining how tasks are to be distributed. To begin a distributed computation session the user sends the task code to be processed to the Hive. The Hive must then decompose the problem into the optimum level of task granularity. Each component process is then *wrapped* i.e. associated with the necessary code for mobility, messaging, and distribution. This code is termed a *Queen* agent. A *Queen* is a mobile agent that is responsible for solving part of the larger problem. The *Queen* will become active and move to the first host of its itinerary. On arrival the *Queen* analyses the local conditions and launches as many *Worker* threads as the local host can comfortably support. The *Worker* threads then begin to solve the task in question. *Scout* agents are periodically created by the *Hive* and dispatched to hosts around the network. On arriving on remote hosts *Scouts* perform tests on the local resources and analyse what hardware and software is present. This information is then passed to the

Hive. The *Queen* also monitors the local host resources (just as the *Scout* does) for indications of activity that would suggest that someone has begun to use the machine again. In this event, the *Queen* will kill its worker threads and leave for the next machine in its itinerary.

Similar to MATS, Traveler uses *information-collection* agents to roam through the network and search for availability of resources. Traveler provides a mechanism for clients to wrap their parallel applications as mobile agents. The agents are dispatched to a resource broker who forms a parallel virtual machine atop available servers to execute the agents based on knowledge provided by the *information-collection* agents. Each agent that represents a client's task is cloned among the servers that form a virtual machine to accomplish their task by collaboration. Throughout the lifetime of an agent, availability of the computational resources of its servers may change with time, but the virtual machine is restricted to be re-configured (i.e. add/remove/change a server) only in between phases of a computation.

FLASH (Flexible Agent System for Heterogeneous Cluster)[65][81][107] offers a framework for building distributed, load-balanced applications in a heterogeneous environment. In FLASH, a *system* agent maintains information of the whole system and passes it to *node* agents of each host on the network, which keep information about the locally residing mobile agents. The sub-tasks of a parallel application are represented by *user* agents i.e. mobile agents which are responsible for migrating though the cluster searching for free resources. Their migration decisions are based on internal states as well as internal and external events. FLASH combines system and application-integrated load management in a single environment. Therefore it is able to react efficiently to changing dynamic background load and avoids unnecessary migration of agents with a short life span, since *user* agents can take application information into account for the migration decisions. However, the standard FLASH environment uses only one system agent[50], which inherits the disadvantages of a centralised scheme. If a developer needs to use a distributed scheme of FLASH, s/he has to implement by him/her self special functional modules to support the distribution of information between the multiple *system* agents and also define the interaction between them. In conclusion, it should be noticed that FLASH is a flexible environment for distributed applications with no fixed mechanisms which allows to experiment with several LB strategies e.g. toggle agents' migration ability, consider or not the server's utilisation load.

3.3.3.3. Model-based load balancing

Almost all the systems that explore the model-based approach to load balancing, use distributions of CPU load and expected process lifetime to decide if and when to migrate. Malone's Enterprise[98] utilises a market mechanism to support load balancing. Instead of using money, agents make bids giving the estimated time to complete a job. In contrast, the Challenger[28] system optimises the load by introducing learning behaviour in the bidding agents to deal with important parameters which have a major impact on system performance – such as message delay and error in estimating the job's completion time. In Eager et al.[48] model-based load balancing processors do time-sharing i.e. they run more than one task at a time, in contrast with Enterprise and Challenger systems, which assume that processors are resources which can only be utilised by one task at a time. However, Eager's algorithm suffers from the same shortcomings as Enterprise in that it cannot adapt, and is thus not robust under a wide range of operating conditions.

Other approaches like [78][131][134] achieve load balancing without estimating process lifetimes. For instance, the TAGS algorithm presented in [78] works as follows. Given a collection of servers $S_1,S_2,S_3,\ldots S_h$ where $S_1<S_2<S_3\ldots<S_h$ (i.e. the processing power of S_2 is greater than S_1) all incoming jobs are immediately dispatched to S_1. There they are served in FCFS (First Come First Serve) order. If they complete before using up S_1's processing power, they simply leave the system. However if a job has used S_1's amount of CPU and still has not completed, then it is killed. The job is then placed at the end of the queue at S_2, where it is restarted from scratch. Each server executes the jobs in its queue in FCFS order. If a job at S_i uses up its amount of CPU and still has not completed it is killed and placed at the end of the queue for S_{i+1}. Although the algorithm *wastes* a large amount of resources by killing jobs and then restarting them from the scratch, comparisons have shown that it outperforms the random, round-robin and central-queue policies of distributing tasks.

In the random assignment policy, an incoming job is sent to S*i* with probability $1/i$, where i is the number of servers. In round-robin assignment, jobs are assigned to servers in a cyclical fashion with the h^{th} job being assigned to server h mod i, where the central-queue policy holds all jobs at the dispatcher in a FCFS queue and only when a server is free does the server request the next job. In [79] it is shown that random and round-robin have almost identical performance. A different approach which is also not

based on estimation of process lifetimes but on local rate of change observations i.e. decision to initiate load transfers do not depend on a server's load but on how the load changes in time, can be found in [132].

3.3.3.4. Other load balancing approaches

Other approaches to load balancing exist, where instead of having a central component deciding when and where to launch processes, human users decide if they want to provide their resources. OCEAN is a major ongoing project at the University of Florida's CISE department to develop a fully functional infrastructure supporting the automated, commercial buying and selling of dynamic distributed computing resources over the internet. The idea is that anyone with spare cycles should be able to deploy an OCEAN server which can run other people's computing tasks for profit, and any developer should be able to easily write a distributed application which any user with a credit card number (or other means of automatic payment) should be able to deploy in distributed fashion using as many suitable OCEAN servers as they can afford to rent for their particular purpose. OCEAN will likely use a distributed, peer-to-peer double-auction mechanism to ensure that jobs are automatically contracted out to the cheapest suitable available bidders, and that OCEAN servers automatically contract themselves out to run the highest-paying available jobs.

Distributed.net[44] is an example of an organisation which encompass thousands of users around the world, resulting in a parallel computing power of more than 160,000 hosts. The users that wish to volunteer their computer's idle time just have to download an applet in order to share their resources. Other systems which exploit resources of voluntary users are Bayanihan[130], Javelin[13], ATLAS[12], IceT[73].

3.4. Conclusion

This chapter covered both the theory and related work of this research. It described the fundamental concepts for the realisation of a Digital Library and presented the different approaches to address System Integration and Data Management in the context of a DL, where emphasis has been given to the agent technology. The development of agent-based applications lead towards the need of a standard to enable

interoperability between agents on different types of platforms. A review on current standardisation efforts towards this need has been presented; in particular FIPA standard has been described in detail. The importance of management within a MAS with reference to load balance has also been stated. Finally, previous work related to the scope of this research has been reported. Different approaches, methodologies, techniques and projects have been discussed in the context of digital libraries, the compatibility between heterogeneous agent-based systems and the load balancing of mobile agents within a MAS. The three subsequent chapters present the proposed architecture of the SARA active DL, an alternative way of enabling the multi-agent system utilising the DL to interoperate with foreign FIPA-compliant agent-based systems and the management of its mobile agents within the system.

Chapter 4. The multi-agent architecture of the SARA active DL

4.1. Introduction

This chapter presents the agent-based architecture of the SARA active digital library. It defines the positions and roles of each agent within the system and demonstrates how it is possible for the agent communication language and metadata to be encoded in XML. Finally, it discusses how System Integration and Data Management are addressed in the content of SARA MAS based on agent collaboration

4.2. The mobile agent-based architecture for the SARA active digital library

The SARA architecture is composed of a collection of *information* and *web*-servers, each of them having a group of agents. The information-servers have the required computational resources and data repositories to constitute the SARA Digital Library - where the data repositories generally contain pre-processed images or geospatial data about a given region - and support Local Interface Agents (LIA).

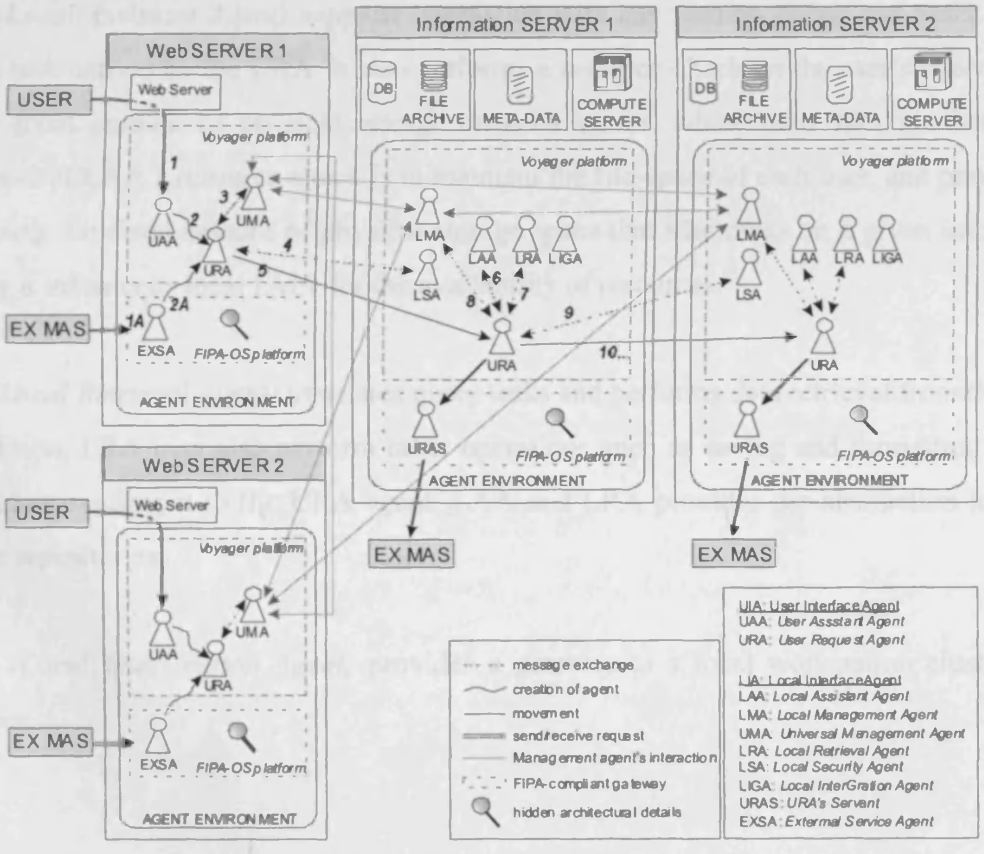


Figure 4.1. The SARA agent-based architecture

The web-servers acts as the front-end to the users that need to access the SARA DL and support User Interface Agents (UIA). Figure 4.1 represents the SARA agent-based architecture and the interaction of the agents between themselves.

Each agent undertakes a particular role in the system. The most complex functionality is localised in non-mobile agents, which remain at one location, providing resources and facilities to lightweight mobile agents that require less processor time to be serialised, and are therefore quicker to transmit. The primary motivations for using mobile agents in this context are: (1) The avoidance of large data transfers - of the order of Terabytes, consisting of sometimes proprietary data, (2) the ability to transfer user developed analysis algorithms, and (3) the ability to utilise specialised parallel libraries.

LIAs are stationary agents that provide an extensible set of services. LIAs provide a level of abstraction between information-servers and the requesting mobile agents, namely:

- *LAA (Local Assistant Agent)* supports interaction with any visiting URAs and assist the completion of the task carried by the URA. It also performs a resource-check on the user's file-space. Each user has a fixed amount of physical storage on each server, where their files are being stored. The objective of LAA's resource-check is to maintain the file-space of each user, and prevent a user from exceeding the fixed amount of physical storage space that s/he owns on a given information-server. Finally it informs its local LMA for the availability of resources.

- *LRA (Local Retrieval Agent)* translates query tasks and performs data retrieval from the local archive. In addition, LRA may also perform other operations such as saving and formatting the results to a file before sending it to the URA agent. LAA and LRA provides the abstraction layer of the data source repositories.

- *LIGA (Local InterGration Agent)* provides a gateway to a local workstation cluster or a parallel machine.

- *LSA (Local Security Agent)* is responsible for authenticating and performing a validation check on the incoming URAs. The URA will be allocated a permission level. Agents from registered users may have access to more information resources than the agents from unregistered users.

- *LMA (Local Management Agent)* coordinates access to other LIAs and supports negotiation among agents. It is responsible for optimizing mobile agents' itineraries to minimise the bottlenecks inherent in parallel processing and ensuring that the URA is transferred successfully. It also informs the rest of the LMAs/UMAs about the status of its local server.

- *UMA (Universal Management Agent).* Similar to LMA, its task is to optimize the overall system's performance. Based on its interaction with each LMA, it is capable of optimising mobile agent migration from the beginning, applying cache techniques, and balancing the distribution of agents between the information-servers. This is due to its information concerning the system status i.e. the status of each server, the availability of resources, the distribution of agents on the network and their activities, any conflicts/failures or updates taking place on the system.

- *URAS (URA's Servant)* is the FIPA-compliant gateway agent of each information-server. Its task is to provide interoperability between the SARA system and a FIPA-compliant one.

- *EXSA (External Service Agent)* is the FIPA-compliant agent of each web-server. Its task is to provide interoperability between a FIPA-compliant system and SARA.

UIAs provide a front-end to the user, for checking/validating their input and displaying the results, namely:

- *UAA (User Assistant Agent)* manages the information of the user and provides control functions for him/her. It launches URAs on behalf of the user, tracks their progress and location, and provides the dispatched URA with a contact point to which the results can be returned i.e. an asynchronous message, if not displayed on a web-page. It also enables the visualisation of results according to the

user's choice e.g. by using different XSL (eXtensible Style-sheet Language) documents to present the results encoded in XML.

- *URA (User Request Agent)* is responsible for migrating to the appropriate local archive site(s), interacting with LIAs at each remote site visited, and returning the results of the user's query to the UAA. If the user's query is broken down into multiple sub-queries, the collaboration of URA with LIGA is necessary to combine the results into a single result that answer the user's query.

The collaboration of the UMAs and LMAs forms a group of agents that contributes to the management of the rest of the agents within the MAS and is discussed in more detail in *Chapter 6*; whereas the alliance of URAS and EXSA contributes to the interoperability of the MAS, discussed in *Chapter 5*. The agents that have been described in this section are compulsory components for the efficient functionality of the SARA architecture. However, the flexibility of the SARA architecture makes it possible for other agents to be engaged in the system to perform other specialised tasks. For instance, a User Profile Agent (UPA) – a type of UIA – can be created to manage the profile of a user and publish useful information for use by the appropriate agents.

The following two sections describe how System Integration (SI) and Data Management is achieved in the content of the SARA multi-agent based architecture.

4.2.1. SI in SARA architecture

As discussed in *Chapter 3 - Background and related work*, SI is important for combining content from different heterogeneous sources. In the SARA architecture, the heterogeneous sources are represented by the information-servers with the required computational resources and data repositories that constitute the SARA Digital Library. The agents described in the previous section provide the necessary functionality to support SI.

UAA stands both as a *mediator* agent that accepts the user's query and translates it into the common model, see example in Code 4.3; and as an *interface* agent that presents the results to the user. The URA is a kind of a *mediator* agent – probably the most important agent – that holds the user's query, carrying

it to the appropriate information sources and interacting with the *source* agents. UMA and LMA are *management* agents that direct the URA as to which server it should migrate to according to its request. Therefore, URA does not need to have the knowledge of the source and its schema to determine which source provides what information. LAA and LRA are identified as *source* agents since their function is to interact with its corresponding information-servers, converting *mediator* queries represented in the common model into queries native to the source and vice-versa. LIGA is another *mediator* agent that is capable of breaking up a query into smaller sub-queries, assigning them to different *mediators* (URAs) and upon receipt of results, combining and integrating these results to form the complete answer to the original query. Finally, EXSA and URAS act as *wrapper* agents to provide interoperability between any external MAS with SARA DL and vice versa; more specifically, EXSA can be considered as an *interface* agent since it provides a gateway to the SARA DL for any external MAS, whereas URAS can be viewed as a *mediator* agent since it is used to retrieve information from external MAS(s).

In order to resolve heterogeneity in terms and definitions used among the agents an ontology is necessary to be defined and is described in *section 4.3* of this chapter. Below, an interaction scheme of the SARA agents based on a simple example of user query processing is described.

4.2.1.1. Agent collaboration support mechanism

This sub-section describes a simple example of user query processing with reference to the SARA agent-based architecture depicted in Figure 4.1. The steps identified in this example are also depicted in the figure of the SARA architecture, for the reader to follow the movement of the mobile agent and its interaction with the rest of the agents. Agent collaboration is also depicted in a sequence diagram in Figure 4.2. The process of agent execution is as follows:

step 1: The user visits the SARA web-server where s/he enters his/her information i.e. the desired query, username, password. The information is gathered by the UAA agent, which is in the form of a servlet.

step 2: The UAA launches a URA by supplying it with the user's information.

step 3: The URA communicates with the UMA which is responsible of constructing the URA's itinerary according to the information provided by the former and the current status of the

system (known to the UMA), i.e. availability of resources, server failures, number of agents on each server. The UMA may also direct the URA to collect the results of its query from a server where they have already been stored by a previous agent having a similar query (see 'Case 1' of *section 6.3.2.6*).

step 4: Once the URA's itinerary is constructed, it communicates with the LSA of the first server of its itinerary.

step 5: After the URA is authenticated and accepted by the server that it needs to migrate to (through the LSA), it migrates to it.

step 6,7: The URA interacts with LAA and LRA which act as wrappers, wraps up the information source and make it thus accessible in a standard form. For instance, the LAA connects to the server's database using JDBC (Java Data-Base Connectivity), then the LRA executes the URA's query and converts the results into XML. Finally, the results are send back to the URA.

step 8: The URA reports its activities on the local server to the LMA. If the URA needs to migrate again and there is a change in the systems status that affects the URA's task, the LMA is responsible of informing the URA and amending its itinerary.

step 9: As in step 4, before the URA migrates to the next server of its itinerary, it needs first to communicate with the LSA of that server.

step 10: Once the LSA has granted access to the URA, the URA moves to the foreign server to continue its task. When the URA accomplish it task, it sends a URL (Uniform Resource Locator) reference with the results of analysis to the UAA. The UAA is then able of presenting the results to the user. Before the URA is self-terminated it also reports to the UMA (from where it was initially launched) details of its task

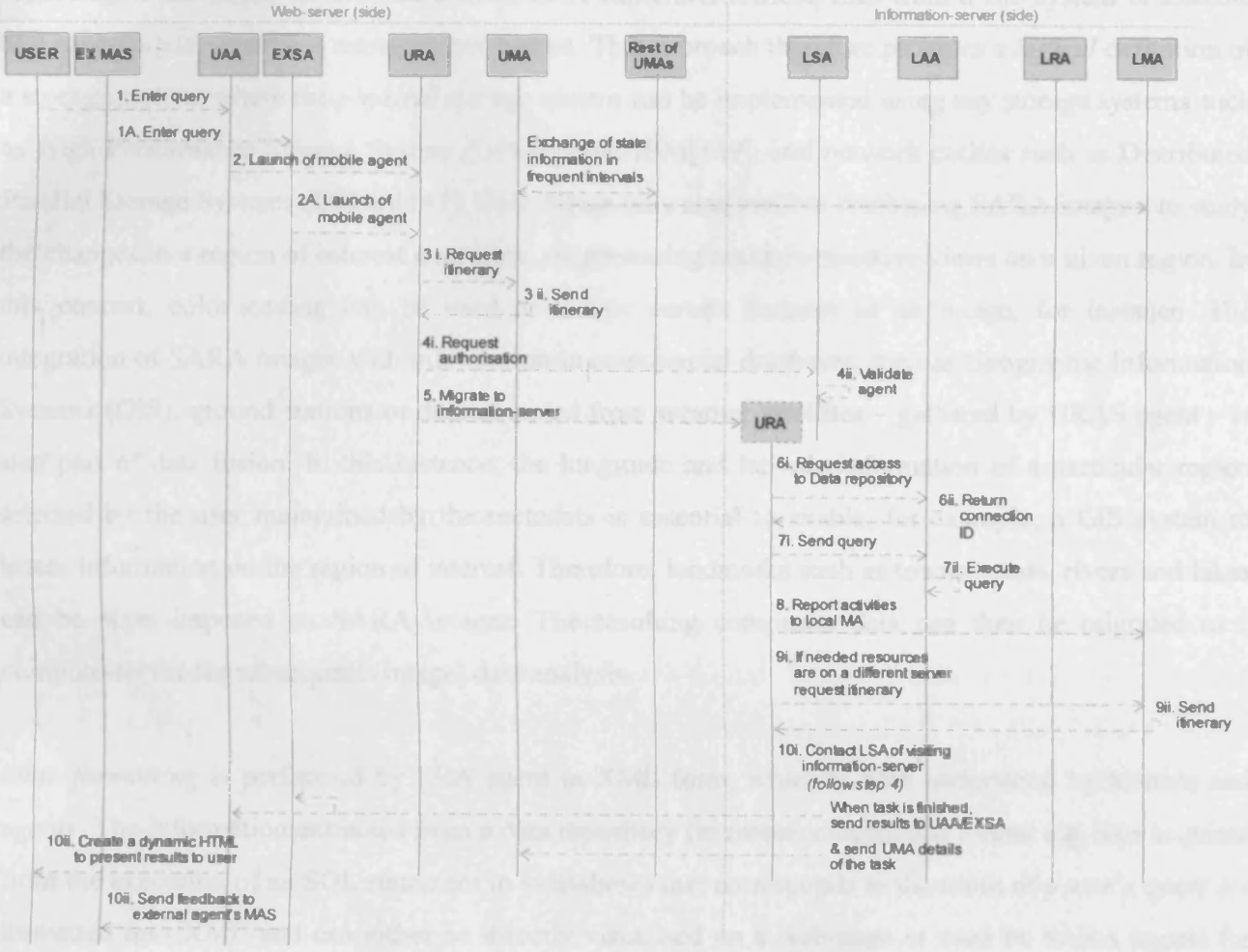


Figure 4.2. Sequence diagram of agent collaboration

4.2.2. Data management in SARA architecture

While *System Integration* deals with the integration of different heterogeneous sources, *Data Management* is important for providing support for managing vast quantities of data representing each digital object to transform them into useful knowledge, see *Chapter 3 - section 3.2.1.2*. The Data Management in SARA architecture is related with the acquisition and representation of data, rather than data entry i.e *data pre-processing*.

Data fusion is supported by the collaboration of LAA and LRA agents. Every SARA information-server has its own LAA that abstracts the type of data source that contains the data, with the data source ranging from flat files to structured databases and its function is to instruct LRA of how to access the data

according to the nature of the data source. LRA can either retrieve files from a file system or execute SQL queries via a database management system. This approach therefore provides a *logical* definition of a storage system, where the *physical* storage system can be implemented using any storage systems such as High Performance Storage System (HPSS) from IBM[149], and network caches such as Distributed Parallel Storage Systems (DPSS)[147]. Data fusion may also involve combining SARA images, to study the changes in a region of interest over time, or generating multi-perspective views on a given region. In this context, color coding can be used to isolate certain features of an image, for instance. The integration of SARA images with information in commercial databases, such as Geographic Information Systems (GIS), ground stations or data obtained from weather satellites – gathered by URAS agent – is also part of data fusion. In this instance, the longitude and latitude information of a particular region selected by the user maintained by the metadata is essential to enable, for example, a GIS system to locate information on the region of interest. Therefore, landmarks such as towns, roads, rivers and lakes can be super-imposed on SARA images. The resulting composite data can then be migrated to a compute-server for subsequent (image) data analysis.

Data formatting is performed by LRA agent in XML form, which is both understood by humans and agents. The information extracted from a data repository (in resource-depended format e.g. data acquired from the execution of an SQL statement in a database) that corresponds to the result of a user's query are formatted into XML and can either be directly visualised on a web-page or used be SARA agents for further processing. An example of such an XML document can be found in Code 4.6 (*section 4.4*) of this chapter.

Data storage in SARA architecture does not regard the entry of data into the digital library, but the storage of data results gathered by LRA and the migration of existing data (performed by URA) to a different information or compute-server, for further data fusion or analysis. In the content of data storage, LAA is responsible of performing a resource-check to maintain the file-space of each user, and prevent a user from exceeding the fixed amount of physical storage space that s/he owns on a given information-server, see *Chapter 7 - section 7.2.1.1.1.*

Data analysis consists of simple activities such as image processing – provided by LAA (developed in SARA prototype, see *Chapter 7 - section 7.2.1.1*) to more compute-intensive tasks ranging from evolutionary computing approaches such as neural networks and genetic algorithms, rule based approaches based on predicate/propositional logic to Case Based Reasoning (CBR) systems, to statistical approaches such as regression – provided by legacy software residing in the compute-servers. The SARA architecture supports *on-demand processing*, meaning that a data archive is connected to a powerful compute-server at high bandwidth, controlled by a user who may be connected at low bandwidth. Therefore it is possible for a user to control and manage the compute-servers via UAA using the SARA GUI (Graphical User Interface). Moreover, the agent-based approach provides additional flexibility in enabling a user to develop his/her own analysis algorithms and transfer them to compute-servers for local processing. This is achieved by attaching a user's custom algorithm to its representative URA agent along with its query. The ability of SARA architecture to provide computing services in addition to data-retrieval services, so that users can initiate computing jobs on remote supercomputers for processing, mining, and filtering of the data, characterises the SARA digital library as *active*; since users can process available data not just to retrieve a particular piece of information, but to infer new knowledge about the data at hand. Finally, the results can either be visualised (*Data visualisation*) using a Web-browser managed by a UAA, or via a more sophisticated immersive environment, such as a CAVE[25].

4.3. Agent communication language

The SARA DL uses an XML schema for agent communication, that combines the most attractive aspects of KQML, FIPA ACL and other agent communication languages, and that enables agents to communicate with each other by expressing intentions in the SARA ontology. The XML schema allows efficient parsing and is modular and flexible to support evolving classes of XML documents. In addition, it retains its simplicity and clarity, and is readable by the user. Each message has a standard structure showing the message type, context information and the body of the message; see Code 4.1.

Message *type* represents intentions such as request, response, failure and refuse explicitly. For instance, a message can be defined for a request to search for tracks, and another message for information passing to return tracks. *Context* is used to identify the sender, the intended recipient of the message or originator for forwarded messages, using some form of local, regional or global naming scheme. *Returnby* sets a

deadline for a users waiting time. *Content* defines the itinerary of the mobile agent, the user's request wrapping in XML, and a recipient or physical location (i.e. a user's directory) to return/save the result.

Autonomous agents co-operate by sending messages and using concepts from the SARA ontology. In the SARA DL, the ontology describes terms and concepts (such as a Track, Latitude/Longitude coordinates, etc) and their inter-relationships. The agent sending and receiving the message must share an understanding of what the words are intended to mean. The system ontology is presented by listing terms, their meanings and intended use in the Document Type Definition (DTD). Although we are aware that an ontology is a much more complex representation of terms and relationships (primarily to support reasoning on these terms), we use a DTD as a simple instance of an ontology. We believe a DTD is adequate for this application - although a more complex representation may be useful in future. Every specific XML specification is based on a separate DTD that defines the names of tags, their structure and concept model. A DTD can determine elements, attributes, types, and required, optional or default values for those attributes. While the XML specification constraints the structured information, the DTD defines the semantics of that structure. An example of an XML message exchange between the UAA and URA agent is shown below.

```
<?xml version="1.0" ?>
<!DOCTYPE message SYSTEM "message.dtd">
<Message type="request" id="USERID">
  <Context sender="//gallium_8000_chris_1032963326234_uaa"
           receiver="//gallium_8000_chris_1032963326234_ura"
           returnby="11/06/04 5pm" />
  <Content>
    <itinerary>
      <server>131.251.42.21:8000</server>
      <server>131.251.42.203:8000</server>
    </itinerary>
    <querydef>
      &trackq;
    </querydef>
    <results>geolos@cs.cf.ac.uk</results>
  </Content>
</Message>
```

introduced by UMA/LMA afterwards

Code 4.1. UAA-URA message exchange in XML format

The DTD of the above XML document and other types of agent message exchange like a *response, failure, refuse* is represented in Code 4.2. The DTD specifies all of the legal message types, constraints on the attributes and message sequences. The *trackq* pointed by the arrow in Code 4.1 represents the

user's request transformed into XML format; see example in Code 4.3, the message's DTD is depicted in Code 4.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Message (context+, content+)>
<!ATTLIST Message type (request|response|failure|refuse) #REQUIRED
   data CDATA #IMPLIED
   id CDATA #REQUIRED>
<!ATTLIST context
   sender CDATA #IMPLIED
   receiver CDATA #IMPLIED
   originator CDATA #IMPLIED
   returnby CDATA #IMPLIED>
<!ELEMENT content (itinerary+, querydef+, results)>
<!ELEMENT itinerary (server)+>
<!ELEMENT server (#PCDATA)>
<!ENTITY trackq SYSTEM "trackquery.xml">
<!ELEMENT querydef (#PCDATA)>
<!ELEMENT results (#PCDATA)>
```

Code 4.2. The DTD for describing XML agent exchange messages *(message.dtd)*

```
<?xml version="1.0" ?>
<!DOCTYPE trackquery SYSTEM "trackquery.dtd">
<trackquery>
  <Condition>
    <and>
      <MoreThanOrEqual>
          <left>latitude.upperleft</left>
          <right>33.132</right>
      </MoreThanOrEqual>
      <MoreThanOrEqual>
          <left>longtitude.upperleft</left>
          <right>-115.196</right>
      </MoreThanOrEqual>
      <MoreThanOrEqual>
          <left>latitude.upperleft</left>
          <right>33.501</right>
      </MoreThanOrEqual>
      <LessThanOrEqual>
          <left>longtitude.upperleft</left>
          <right>-114.607</right>
      </LessThanOrEqual>
      <LessThanOrEqual>
          <left>latitude.upperleft</left>
          <right>32.775</right>
      </LessThanOrEqual>
      <MoreThanOrEqual>
          <left>longtitude.upperleft</left>
          <right>-113.969</right>
      </MoreThanOrEqual>
      <LessThanOrEqual>
          <left>latitude.upperleft</left>
          <right>32.409</right>
      </LessThanOrEqual>
      <LessThanOrEqual>
          <left>longtitude.upperleft</left>
          <right>-114.555</right>
      </LessThanOrEqual>
    </and>
  </Condition>
</trackquery>
```

Code 4.3. An example of a user's request encoded in XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT trackquery (condition+)>
<!ELEMENT condition (and|or)+>
<!ELEMENT and (Equal|LessThanOrEqual|MoreThanOrEqual)+>
<!ELEMENT or (Equal|LessThanOrEqual|MoreThanOrEqual)+>
<!ELEMENT Equal (left,right)>
<!ELEMENT LessThanOrEqual (left,right)>
<!ELEMENT MoreThanOrEqual (left,right)>
<!ELEMENT left (#PCDATA)>
<!ELEMENT right (#PCDATA)>
```

Code 4.4. The DTD for describing the user's request *(trackquery.dtd)*

In XML-based messages, agents encode information with meaningful structure and commonly agreed semantics. On the receiving side, different parts of the information can be identified and used by different services. XML may also be used to provide a means for agents to express their beliefs, desires and intentions based on a BDI model. Moreover, a mobile agent can carry an XML document to a remote data archive for data exchange, where both queries and results are XML-encoded.

4.4. XML-based data specifications

In choosing how the SARA system exchanges results trade-offs need to be considered, for example between efficiency and flexibility. The efficiency of the communication is maximised by bulk binary transfers where sender and receiver know everything about the transfer before it begins. Flexibility is maximized by using ASCII text with redundancy and syntax checking.

In the SARA system, XML is used to encode system structure as metadata. The Entity Attribute Relationship model (EAR) for the metadata is shown in Figure 4.3. The metadata consists of four tables. The *Track* table contains information about the images i.e. their name, date of acquisition, unique ID, width, height and number of channels. A channel can be perceived as an alternative visualisation version of the original image based on the way of its acquisition from the SIR-C shuttle, for further information see [153]. The *Coords* table contains the latitude and longitude coordinates of the four vertices for each image. The *File* table contains filenames constituting the images, and finally the *Stored* table contains information about where the images are actually stored.

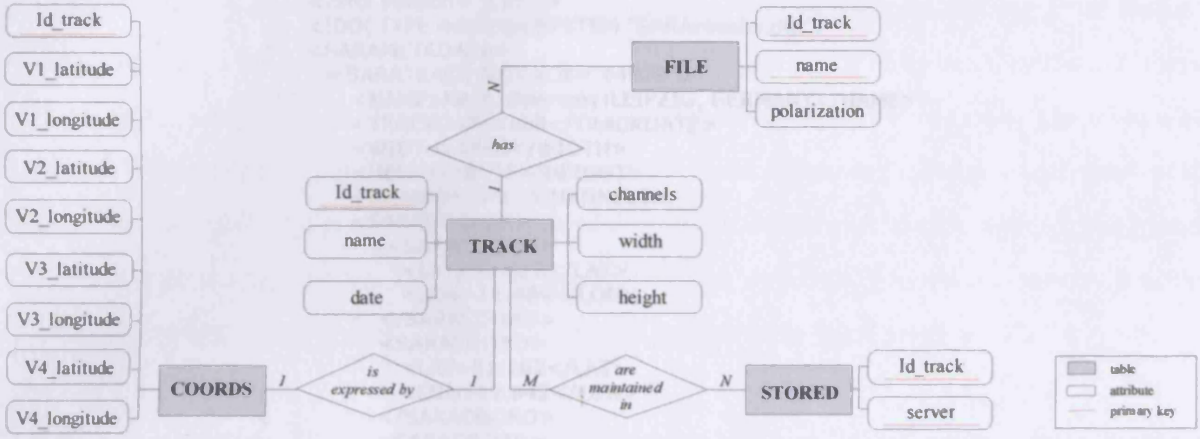


Figure 4.3. Entity Attribute Relational model (EAR) for the SARA metadata

The DTD is presented below in Code 4.5. A demonstration of an XML document produced according to the DTD can be found in Code 4.6.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT CHANNELS (#PCDATA)>
<!ELEMENT HEIGHT (#PCDATA)>
<!ELEMENT LAT (#PCDATA)>
<!ELEMENT LON (#PCDATA)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT POLARIZATION (#PCDATA)>
<!ELEMENT SARACOORD (LAT,LON)>
<!ELEMENT SARACOORDS (SARACOORD+)>
<!ELEMENT SARAFILE (POLARIZATION)>
<!ATTLIST SARAFILE NAME ID #REQUIRED>
<!ELEMENT SARAFILES (SARAFILE+)>
<!ELEMENT SARAMETADATA (SARATRACK+)>
<!ELEMENT SARASTORED (SERVER+)>
<!ELEMENT SARATRACK(NAME,TRACKDATE,WIDTH,HEIGHT,CHANNELS,
          SARACOORDS,SARAFILES, SARASTORED)>
<!ATTLIST SARATRACK IDTRACK ID #REQUIRED>
<!ELEMENT SERVER (#PCDATA)>
<!ELEMENT TRACKDATE (#PCDATA)>
<!ELEMENT WIDTH (#PCDATA)>
```

Code 4.5. The DTD for describing the SARA metadata *(SARAresults.dtd)*

```
<?xml version="1.0" ?>
<!DOCTYPE message SYSTEM "SARAreusIts.dtd">
<SARAMETADATA>
  <SARATRACK IDTRACK="44829">
    <NAME>Harz, Germany:LEIPZIG, GERMANY</NAME>
    <TRACKDATE>null</TRACKDATE>
    <WIDTH>1448</WIDTH>
    <HEIGHT>8555</HEIGHT>
    <CHANNELS>2</CHANNELS>
    <SARACOORDS>
      <SARACOORD>
        <LAT>51.628</LAT>
        <LON>11.484</LON>
      </SARACOORD>
      <SARACOORD>
        <LAT>51.182</LAT>
        <LON>12.842</LON>
      </SARACOORD>
      <SARACOORD>
        <LAT>51.037</LAT>
        <LON>12.719</LON>
      </SARACOORD>
      <SARACOORD>
        <LAT>51.481</LAT>
        <LON>11.364</LON>
      </SARACOORD>
    </SARACOORDS>
    <SARAFILES>
      <SARAFILE NAME="pr44830_byt_hv">
        <POLARIZATION>CHV</POLARIZATION>
      </SARAFILE>
      <SARAFILE NAME="pr44830_byt_vh">
        <POLARIZATION>CVH</POLARIZATION>
      </SARAFILE>
    </SARAFILES>
    <SARASTORED>
      <SERVER>server1</SERVER>
    </SARASTORED>
  </SARATRACK>
</SARAMETADATA>
```

Code 4.6. An example of an XML document representing SARA data

4.5. Properties of the SARA architecture

The SARA architecture has been designed to fulfill the following properties:

- *Modularity*. The system component is composed of interchangeable modules, each providing some of the required functionality. For instance, Local Assistance Agent (LAA) abstracts the type of data source that contains the data, with the data source ranging from flat files to structured databases. If the local archive system changes, the only agent that will need to be amended is the LAA.

- *Scalability.* The system component deals with a large number of requests coming from many users simultaneously. As the load grows, the system should scale gracefully. The collaboration of the UMA and the LMA intelligently distributes the mobile agents among the servers and assigns computational resources. The effect of the Management Agents on load balancing as the agent load scales is discussed in Chapter 8. In particular, the chart of Figure 8.8 in *Chapter 8 - section 8.2.3* demonstrates the total task time required by 200 agents launched in five different information-servers to complete their tasks in conjunction with two other LB schemes, which came with better results.

- *Semi-Decentralisation.* The system is open and evolving. There is no global administrator agent, because agents are submitted from various remote sites and it would be inefficient to route all agents through a central site. The management of the agents within the SARA architecture is performed by the UMA and the LMA which form a semi-decentralised scheme based on their position in the network.

- *Extensibility/dynamically.* The system component allows new elements, such as new services and new archive systems, to be easily added. A new service may be introduced as an extension of an existing agent's responsibilities or represented by a new generated agent. Database/archive or compute-servers may be dynamically added or removed into/by the system. The only requirement is to inform the management agents about the corresponding update on resources. The other agents do not need to be informed.

- *Reliability/fault-tolerance.* The system performs reliably during network overload, failures and updates. The UMA and the LMA are responsible for informing URAs of any changes and updates taking place in the system. In case of failure, the management agent of the server that crashed can automatically restore its state and re-connect to the system once the problem that caused the crash is recovered e.g. an electrical break-down.

- *Flexibility.* The system is customisable. Users may attach their own filtering algorithms to the URA to be transferred on a remote compute-server for local data fusion and analysis.

- *Security*. The system is secure. The LSA stands as a shield for the system. Only authorized agents are permitted to enter, communicate and perform in the SARA system.

- *Interoperability/compatibility*. The system can interoperate with other FIPA-compliant MAS(s) and vice-versa. In this instance, services and/or information resource of a Geographic Information System (GIS) may be used by SARA users for enhanced data fusion e.g. the longitude and latitude of a particular area of the earth can be used as parameters on a GIS to retrieve land information such as street names, which can then be combined with SARA image(s) of the corresponding geographical coordinates, resulting in a detailed map of the particular area.

4.6. Conclusion

This chapter has proposed an agent-based architecture for the realisation of a digital library with reference to SARA active DL. The advantages of the approach utilising the agent technology have also been stated. The interoperability part of the architecture, as well as the load balancing of mobile agents within the MAS are described in the following two chapters.

Chapter 5. Interoperability of multi-agent systems

5.1. Introduction

The ability to enable a legacy agent-based system utilising a digital library to interoperate with other agent-based systems, extends its capabilities for its users with further services and information resources. In *Chapter 3 - section 3.2.4.1* the most important approaches that define interoperability between agents on different types of platforms have been discussed and compared; FIPA has been distinguished as to the most accepted standard in agent community due to its features. This chapter proposes an alternative approach to conforming an agent-based legacy system to a FIPA-compliant one with the use of gateways, which behave like wrappers between the non-FIPA compliant system and the FIPA-compliant ones. The architecture of the generic FIPA-compliant gateways that could be attached to a legacy MAS provides automated FIPA interoperability with an external FIPA-compliant MAS, saving a developer time in terms of reading, understanding and applying the FIPA specifications to a MAS. The adoption of the FIPA-compliant gateways from SARA MAS for inheriting FIPA compliance is also discussed in this chapter. Note that the legacy code of the SARA prototype was developed before the need to provide an interoperable layer to the system was identified.

5.2. An approach to conforming a MAS into a FIPA-compliant one using FIPA-compliant gateways

The conversion of a MAS into a FIPA-compliant system (i.e. a system that adheres to FIPA standards) implies that system developers must rebuild their systems based on FIPA specifications. Such a conversion imposes amendments on the system architecture to conform to the new standards, which may results in extensive code rewriting and testing. Based on the guidelines provided by FIPA association, for an agent platform implementation to be considered FIPA-compliant, it must at least implement the "Agent Management" and "Agent Communication Language" specifications, which should conform to the latest *experimental* and/or *standard* status specifications.

The usual approach to conforming a MAS into a FIPA-compliant one is to modify the whole system based on FIPA specifications. A different approach that has not yet been adopted by any developer is to amend just a part of the system's architecture. The top picture of Figure 5.1, represents a typical multi-

agent system (MAS 1) that has been conformed to FIPA specifications in order to be able to interoperate i.e. receive/send data from/to other FIPA-compliant multi-agent systems (EXternal MAS). Figure 5.1b, represents the approach proposed in this thesis of how a MAS can be conformed to a FIPA-compliant one. The actual architecture of the system remains the same as before, but two FIPA-compliant gateways (in grey) have to be added to the system. These work as *adaptors* (wrappers) to ensure interoperability with other FIPA-compliant external multi-agent systems (EX MAS). Interoperability in this sense applies at both the communication and application levels. The communication level comprises the connection and communication layer, whereas the application level comprises the ontological and agent service layer[27].

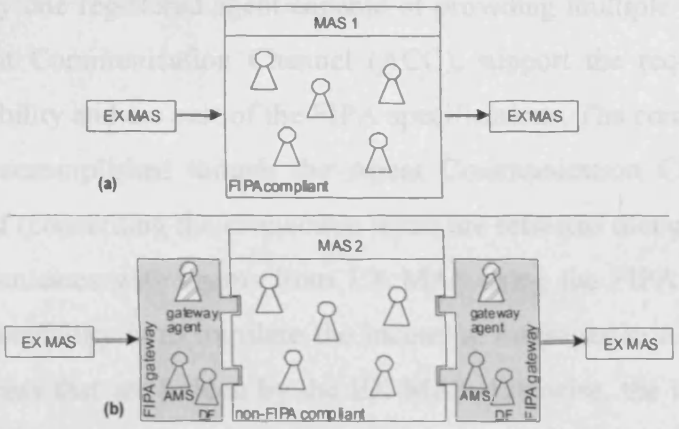


Figure 5.1. Two different approaches of conforming an agent platform into a FIPA-compliant one

Note that this approach should not be confused with agent software integration support of FIPA specifications (see FIPA[54] specification FIPA00012), or with similar approaches that claim FIPA compliance but they actually alter[117] the original FIPA specifications. This approach proposes a novel architecture of generic FIPA-compliant gateways that could be attached to a legacy MAS to provide automated FIPA interoperability with an external MAS. By the term automated it is meant that a developer would not need to have any knowledge of the FIPA specifications in order to make their system FIPA-compliant. For this purpose, a special API written in Java has been created to facilitate the realisation of the FIPA-compliant gateways, and this is explained later on in *section 5.3* of this chapter. Although the proposed architecture of the generic FIPA-compliant gateways supports a limited number

of performatives, a developer would be able to extend the gateway agent Java Class in order to support any performative that it is not initially supported by the generic architecture.

The two gateways are the FIPA-compliant part of the system. Each of those has all of the mandatory, normative components of the FIPA architecture. Each gateway contains three agents: the Agent Management System (AMS), the Directory Facilitator (DF) and the gateway agent. The AMS and DF are the FIPA agents, as defined by FIPA specifications. The gateway agent is the only agent of the system registered by both AMS and DF, which acts as a wrapper between MAS2 and any external MAS. All the available services of the system are represented by this agent. It is like having an ordinary FIPA compliant system with only one registered agent capable of providing multiple services. The Directory Facilitator (DF) and Agent Communication Channel (ACC), support the required infrastructure for enabling service interoperability and are part of the FIPA specifications. The communication between an EX MAS and MAS2 is accomplished though the Agent Communication Channel (ACC) and the protocols that are supported (concerning the connection layer) are reflected thought the platform address. The gateway agent communicates with agents from EX MAS using the FIPA Agent Communication Language (ACL). Its responsibility is to translate the incoming messages to a form understood by its internal agents i.e. the agents that are hidden by the EX MAS. Likewise, the internal agents' requests have to be also converted by the gateway agent into ACL messages, in order to be understood by an EX MAS. The gateway agent maintains a list of the agents within the system being wrapped, along with the registered services (with DF) that each of them can provide. Therefore, based on the service requested by an EX MAS, the gateway agent knows to which system agent the message should be forwarded, after it has been translated into the form understood by the appropriate agent that receives the request.

Hence, the external MAS does not *see* anything else apart from the gateway agent; which on receiving a request from an external MAS (on the left side of MAS2) is responsible for transferring the request to the agents of its system, which are hidden by the external MAS, for processing the request. Once the request is accomplished, a response is returned to the external MAS through the gateway agent. In the case where agents from MAS2 need to communicate with an external MAS (on the right side of MAS2), their request is passed through the gateway agent and translated into ACL; the results gathered by the external MAS are returned to MAS2 agents through the gateway agent as well.

The capability of the FIPA-compliant gateway may be further extended by defining extra sets of operations that may be supported by these agents. For instance, the utilisation of a security layer will enable heterogeneous MAS to interoperate using X.509[168] based digital certificates. In addition, an agent mobility layer would provide the capability to support agent migration between heterogeneous MAS built on the same agent platform.

5.2.1. Supporting multiple gateway agents

Although one of the advantages of the FIPA-compliant gateway is to *isolate* the externally accessible part of the architecture i.e. the gateways, from the rest of the system for increasing security (since the policy of the architecture remains hidden to a foreign Agency), some developers might need to expose more than one agent to an external MAS.

This could be achieved by adding multiple gateway agents to the FIPA-compliant gateway that provides interoperability between the legacy MAS and an external one, as shown in Figure 5.2a. In this case, the agent that would need to be directly accessed by an external MAS could be represented by a separate gateway agent. For instance, with reference to Figure 5.2a, agent1 with service1 is resented by gateway agent1 (GA1), service2 of agent2 by GA2 and service3/4 & 5 by GA3.

Even in the case where all of the available services provided by a legacy MAS are represented by a single gateway agent, the introduction of multiple gateway agents with replicated services in the FIPA-compliant gateway may also be useful for:

- Balancing the incoming requests among the existing gateway agents. In a MAS with numerous received requests, the gateway agent that receives a request from an EX MAS may pass the request to another (less occupied) gateway agent. For instance, the steps that have to be followed in order for a message to be passed from one gateway agent to another one, see Figure 5.2b, are:

  Step 1: An agent from an EX MAS sends a request to GA1.

  Step 2: If the message is not understood by GA1, it replies to the sender agent with a "Not-understood" message, otherwise it sends an "Agree" message including the parameter

"reply-to" with the gateway agent's name to which the message is forwarded i.e. GA2. Therefore, subsequent messages (from the external agent) will be directed to GA2.

Step 3: GA1 forwards the external agent's message to GA2 via an "Inform" message including the parameter "reply-to" with the external agent's name.

Step 4: GA2 communicates with its appropriate internal agent according to the service required. The message that is sent to the internal agent is the content of the GA1's message (sent to GA2), which has already been translated by GA1 (to validate the external agent's message) to the form understood by their internal agents.

Step 5: GA2 upon receipt of results from its internal agent, generates an ACL message and sends it to the external agent via an "Inform" message.

The sequence diagram of Figure 5.3 demonstrates the above agent communication.

- Increasing fault tolerance of the interoperability part of a legacy MAS. The FIPA-compliant gateways may be configured to be distributed i.e. each gateway agent to be distributed on a different host. Therefore, even if one of the gateway agents fails, the MAS may still be able to provide its services to an external MAS through the rest of the gateway agents.

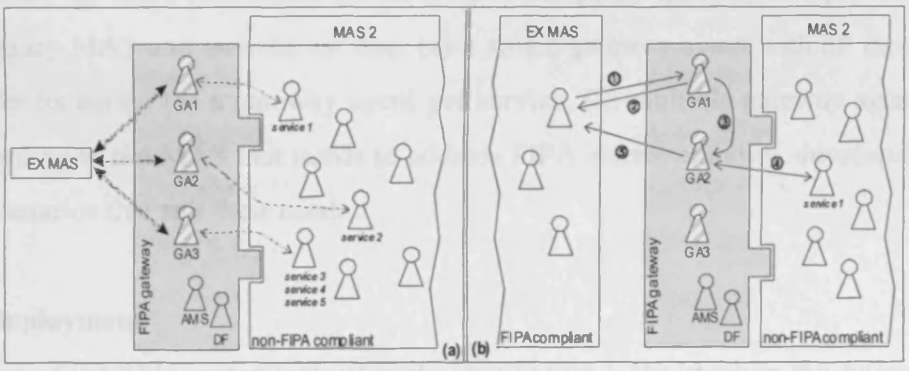


Figure 5.2. Multiple gateway agents

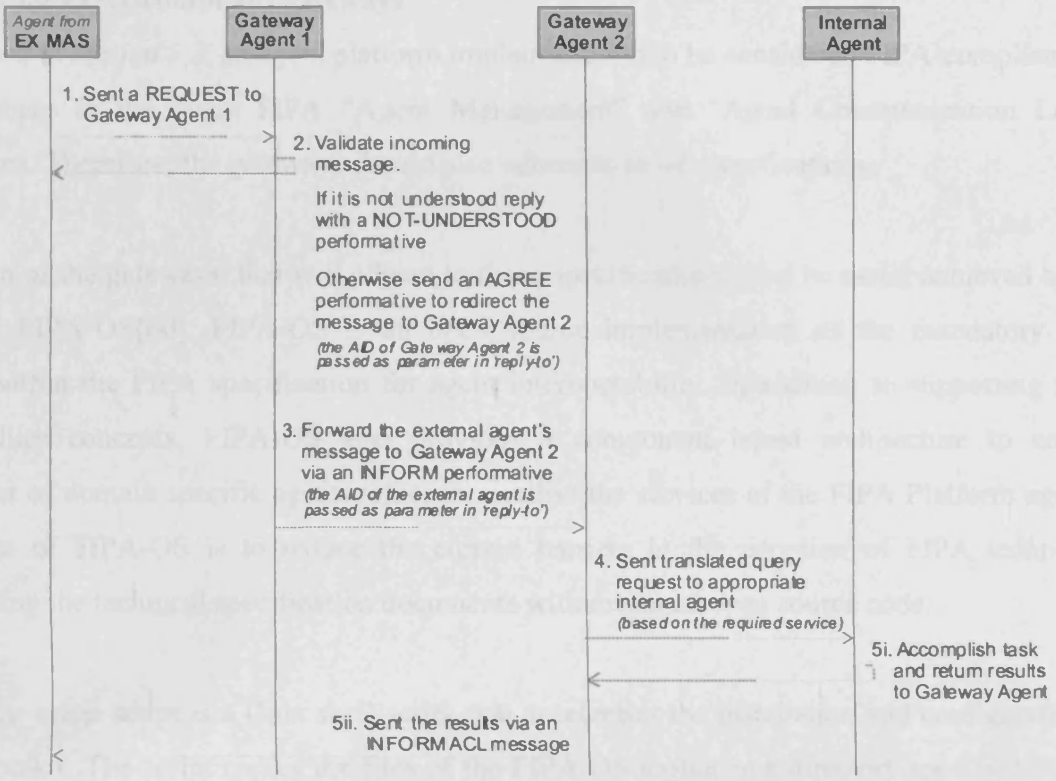


Figure 5.3. REQUEST forwarded to a different Gateway Agent

To conclude, there are three case scenarios for a FIPA-compliant gateway that provides interoperability between the legacy MAS and an external one: (a) a single gateway agent with all the available services registered under its entity, (b) a gateway agent per service, (c) multiple gateway agents with replicated services. According to the MAS that needs to address FIPA interoperability, developers can choose one of the above scenarios that suit their needs.

5.3. Steps of deployment

The deployment of the FIPA-compliant gateways (see Figure 5.1b) involves the following steps: (a) the creation and configuration of the two FIPA-compliant gateways i.e. one to support interoperability between an external MAS and the legacy one, and one, vice-versa, and (b) the creation of each of the gateway agents i.e. one per gateway. To facilitate the realisation of the FIPA-compliant gateways a *gateway_setup* script has been developed for the setup of the gateways, and the *GatewayAgent* (GA) API for the configuration and maintenance of the gateway agents. The following sub-sections describe in detail the realisation of steps (a) and (b).

5.3.1. Creating FIPA-compliant gateways

As mentioned in *section 5.2*, an agent platform implementation to be considered FIPA-compliant, it must at least adhere to the latest FIPA "Agent Management" and "Agent Communication Language" specifications. Therefore, the gateways should also adhere to those specifications.

The creation of the gateways, that will adhere to those specifications, may be easily achieved by using a toolkit like FIPA-OS[60]. FIPA-OS is an open source implementation of the mandatory elements contained within the FIPA specification for agent interoperability. In addition to supporting the FIPA interoperability concepts, FIPA-OS also provides a component based architecture to enable the development of domain specific agents which can utilise the services of the FIPA Platform agents. The primary aim of FIPA-OS is to reduce the current barriers in the adoption of FIPA technology by supplementing the technical specification documents with managed open source code.

The *gateway_setup* script is a Unix shell-script that accelerates the installation and configuration of the FIPA-OS toolkit. The script copies the files of the FIPA-OS toolkit in a directory specified by the user and configures the gateway based on the information inputted by the user during the execution of the script. This information includes a name for the agent platform (gateway) and a list of the external platform-names that the MAS will need to interoperate with. After the FIPA-OS toolkit files have been copied to the destination directory, the script setups the gateway by modifying the configuration parameters of the toolkit stored in XML files - based on the user-input mentioned above. Similar scripts may be developed for other Operating Systems, as long as FIPA-OS toolkit supports them, such as Windows XP. Details on the *gateway_setup* script can be found in *Appendix A2*.

Once the configuration of the toolkit is finished, the execution of a simple FIPA-OS script (named '*startFIPAOS*') starts-up the configured FIPA-agent platform i.e. FIPA-compliant gateway, with the AMS and DF agents initialised. The last piece remaining for the implementation of the FIPA-compliant gateways are the gateway agents.

5.3.1.1. Gateway agent: EX MAS to legacy system

An example of a simple gateway agent written in Java using the GatewayAgent (GA) API is demonstrated below. Actually, the following code example shows the implementation of the SARA gateway agent i.e. EXSA.

```
1  import GatewayAgent.*;
2  ...
3
4  public class EXSA
5  {
6
7   public void initialise()
8   {
9    GatewayAgent EXSA;
10   IEXSA_serv exsa_serv=null;
11
12   try  // get a proxy for that class
13    {
14     exsa_serv=(IEXSA_serv) Namespace.lookup("//localhost:8000/EXSA_serv");
15    }
16   catch(Exception e) {}
17
18   LinkedList properties=new LinkedList();
19   properties.add("EXSA");
20   properties.add("serve_EXMAS");
21   properties.add("EX_SARA_ontology.dtd");
22   properties.add(exsa_serv);
23   properties.add("EXSA_URA");
24
25     // setup the SARA EXSA Gateway agent
26   EXSA=new GatewayAgent("c:/fipaos/profiles/platform.profile","EXSA","SARA");
27   EXSA.addProperty(properties);
28   ...
29  }
30 }
```

Code 5.1. Example code of the SARA EXSA gateway agent

The commands necessary for the configuration and initialisation of the gateway agent are in red. Firstly, the GatewayAgent library must be imported (line 1). In line 9, EXSA is declared as a gateway agent and is constructed in line 26 by calling the constructor of the GatewayAgent with the following parameters: the location of the "platform.profile" i.e the FIPA-OS configuration file which contains information about the FIPA-agent platform (gateway) installed, a unique name for the gateway agent and a name for its owner. Once the gateway agent has been created, it should be configured i.e. be informed of the available services provided by its internal agents. The *addProperty* method (line 27) of the GatewayAgent configures the EXSA agent based on the information provided in the *properties* LinkedList. Every LinkedList that is passed as a parameter to the *addProperty* method should hold information for a single service and its content should contain the following details in order, as declared in lines 18-23, in this example, for the SARA EXSA gateway agent's service:

i) service-name
ii) service-type
iii)service ontology
iv)the internal agent that provides the corresponding service (i.e. its proxy)
v) the internal agent's method that will be called once a request from an external MAS is received by the gateway agent.

It is possible for a gateway agent to provide support for more than one service. This could be achieved by using a separate *addProperty* method for every single service that needs to be registered. Every gateway agent maintains a *property* list which contains detailed information about all the registered services, where GA API supports dynamic service addition, deletion and updating. The successful configuration of a gateway agent involves the automatic registration of itself to the AMS and DF of its platform.

Therefore, the steps of setting-up a gateway agent with the use of the GA API could be achieved within a few lines of code which involve its creation and configuration. The GA API provides multiple methods for its configuration and maintenance which can be found in *Appendix A3*.

At this point the gateway agent is automatically capable of handling the communication with an external FIPA-compliant MAS regarding a request or a cancellation of a prior request received from the later. This is due to the limited performatives supported by the default GA API. The following two sections demonstrate how a default gateway agent (generated using the GA API) handles a request for a service, supported performatives are discussed and means to extend a gateway agent Class for supporting other performatives not initially defined within the GA API are also described.

5.3.1.1.1. Performative handling by the gateway agent

Once the gateway agent receives a request from an external FIPA agent, it locates its appropriate internal agent that can serve the specified request based on the *property* list the gateway agent maintains. The content of the received ACL message is parsed by the gateway agent against the ontology specified by the requested service and if it is valid, the gateway agent forwards it to its internal agent specified by the

requested service's properties. After the request has been accomplished by the service's internal agent representative, the results are sent to the gateway agent. The gateway agent then generates an ACL message which contains the results received by its internal agent and gives feedback to the external FIPA agent that initially placed the request. The interaction of the gateway agent with the external FIPA agent is handled by the REQUEST performative that the GatewayAgent supports. A developer will only have to implement the method of the internal agent that represents the requested service indicated by the service properties, in this case the EXSA_URA method (of the EXSA agent). The method should be of the form:

```
public String EXSA_URA(String do_undo,String message,String convID)
```

This method receives as parameters the content of an ACL message based on the corresponding service ontology, the conversation ID of the external FIPA agent with the gateway agent, and a String of value "do" or "undo". The conversation ID may be used for supporting conversation sessions i.e. to identify whether a request is related with a prior one. The *do_undo* variable stands as a *flag* which indicates whether a REQUEST or a CANCEL performative has been received, with values "do" or "undo" respectively. Therefore, according to the *do_undo* value the method should either carry out (do) or cancel (undo) the task indicated by the *message* variable. Finally, the method should return a String containing the results of the task that has been carried out. Alternatively, positive or negative value should be returned in the case where a task has to be canceled; the return value is determined based on the successful cancellation of the task. The interaction of an external FIPA agent with a gateway agent on a REQUEST performative sent by the former to the later with a valid content message in depicted in Figure 5.4.

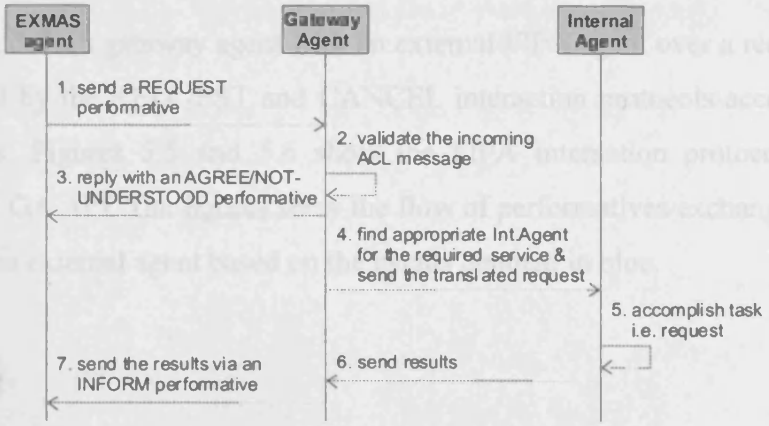


Figure 5.4. Message flow between an external agent and a gateway agent

5.3.1.1.2. Performatives supported by a default gateway agent

The generic FIPA-compliant gateways support a limited number of performatives. A default gateway agent created using the GA API is automatically enabled for handling a request or a cancellation of a prior request received from an external FIPA agent. This involves the support of seven out of the twenty two performatives currently provided by the standard FIPA Communicative Act Library Specification[55], namely:

- AGREE: *"The action of agreeing to perform some action, possibly in the future."*
- CANCEL: *"The action of one agent informing another agent that the first agent no longer has the intention that the second agent performs some action."*
- FAILURE: *"The action of telling another agent that an action was attempted but the attempt failed."*
- INFORM: *"The sender informs the receiver that a given proposition is true."*
- NOT-UNDERSTOOD: *"The sender of the act (for example, i) informs the receiver (for example, j) that it perceived that 'j' performed some action, but that 'i' did not understand what 'j' just did."*
- REFUSE: *"The action of refusing to perform a given action, and explaining the reason for the refusal."*
- REQUEST: *"The sender requests the receiver to perform some action."*

The interaction of a default gateway agent with an external FIPA agent over a request or cancellation of a service is handled by the REQUEST and CANCEL interaction protocols accordingly, as defined by FIPA specifications. Figures 5.5 and 5.6 show the FIPA interaction protocols as they have been implemented by the GA API. The figures show the flow of performatives exchanged between the default gateway agent and an external agent based on the events denoted in blue.

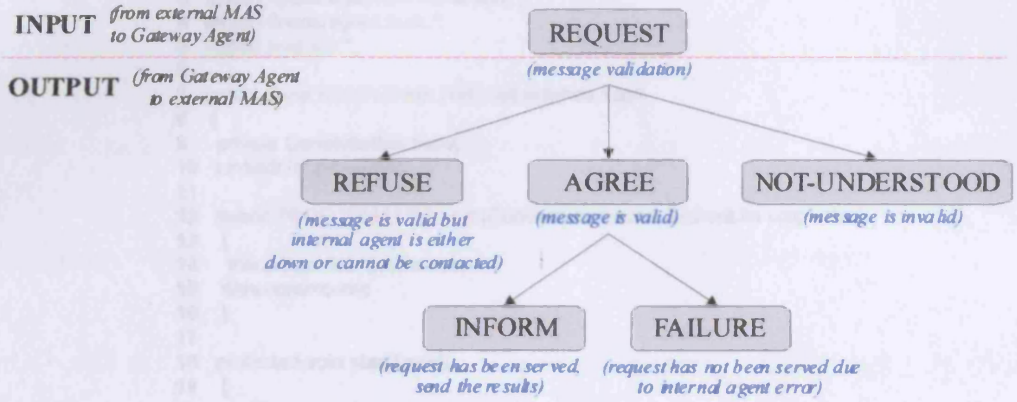


Figure 5.5. FIPA Request interaction protocol

This information does not need to be known by a developer, since s/he does not have to have any knowledge of the FIPA specifications for conforming a legacy MAS to a FIPA compliant one, due to the FIPA-compliant gateways which are by themselves conformed to FIPA specifications.

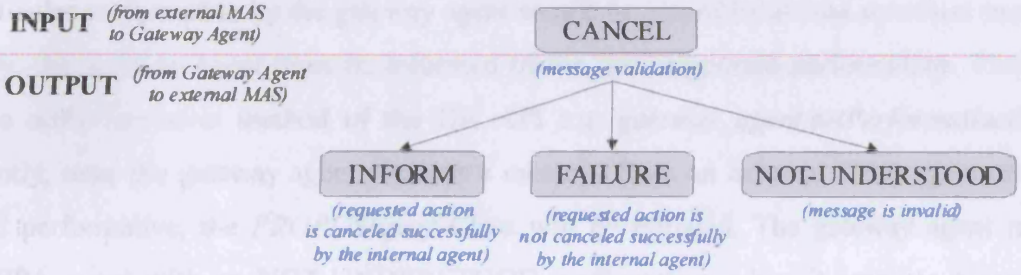


Figure 5.6. FIPA Cancel interaction protocol

A developer is capable of extending the default gateway agent to support other performatives than the ones currently provided by the initial GA API. Based on the GA API, the following Java Class template must be used for any performative that needs to be supported by the default gateway agent.

```
1  package GatewayAgent;
2
3  import fipaos.agent.conversation.*;
4  import fipaos.agent.task.*;
5  import java.util.*;
6
7  public class PERFORMATIVEperf extends Task
8  {
9   private Conversation conv;
10  LinkedList properties;
11
12  public PERFORMATIVEperf (Conversation conv, LinkedList properties)
13  {
14   this.properties=properties;
15   this.conv=conv;
16  }
17
18  protected void startTask()
19  {
20     // developer's code here
21  }
22
23 }
```

Code 5.2. Java Class template of a performative

The Class along with its constructor must have the name of the performative that needs to be supported accompanied by the "perf" string. For instance, to support the PROPOSE performative the name of both the Class and its constructor must be "PROPOSEperf". The code that will be executed upon the receipt of the particular performative by the gateway agent should be placed inside the *startTask* method, in line 20. Finally, the gateway agent must be informed of the new supported performative. This is done by calling the *setPerformative* method of the GA API e.g. *gateway_agent.setPerformative(PROPOSE)*. Consequently, once the gateway agent receives a message from an external FIPA agent containing the PROPOSE performative, the *PROPOSEperf* Class will be initiated. The gateway agent replies to an external FIPA agent with an NOT-UNDERSTOOD performative when it receives a message of an unsupported performative by itself.

The Class template provides the *conv* and *properties* variables (initialised in lines 14 and 15). The *conv* variable contains information regarding a message received from an external FIPA agent like the sender agent-name, the content of the message etc., whereas the *properties* variable contains the gateway agent's *property* list (mentioned in section 5.3.1.1). The *GAparse* method of the GA API may be used to validate the incoming message (included in the *conv* variable), where the FIPA-OS API may be used for structuring and sending a reply ACL message to the external FIPA sender agent.

Extending the default gateway agent requires knowledge of the ACL message structure and the performative specifications that need to be supported by the gateway agent, as specified by FIPA.

5.3.1.2. Gateway agent: legacy system to EX MAS

This gateway agent does not provide any services. Its responsibility is to use services provided by external FIPA compliant MAS(s) on behalf of the agents of the legacy MAS. The implementation of the gateway agent involves its creation (lines 1,9,26 of Code 5.1) and configuration.

The method of the GA API that configures a gateway agent to use service(s) of an external FIPA-compliant MAS is *setEXservices*, see *Appendix A3*. This method receives as parameters information regarding the service(s) provided by an external FIPA-compliant MAS which are intended to be required by the internal agent of a legacy MAS i.e. a list of service names, the external Directory Facilitator's name and a list of the communication protocols supported by the external FIPA-compliant MAS that provides the specified services. For service(s) provided by another external FIPA-compliant MAS, the *setEXservices* method must be called again with input parameters containing information of the corresponding MAS services. The gateway agent maintains a list of all the external services along with their detailed information.

When an agent from a legacy MAS needs to use an external service, its request is passed to the gateway agent by calling the *sendRequest* method of the GA API. The gateway agent is then responsible for making contact and handling the communication with the external FIPA agent that provides the particular service specified by its internal agent to accomplish the request. The communication protocol used by the gateway agent with the corresponding external FIPA agent is defined in the gateway agent's configuration details, where the name of the external FIPA agent that represents the requested service is traced by the gateway agent. This is done by interrogating the external DF about which agent handles the particular service. The *sendRequest* method returns the results of the service requested by an internal agent, where a *not-understood* or a *failed* message-string is returned if the internal agent's request has not been understood or refused/failed to be accomplished by the external FIPA agent.

Note that an internal agent's request (before it is forwarded to an external FIPA agent) has to be translated to the form understood by the latter based on its service ontology, similarly results received from a service provided by an external FIPA agent have to be translated into the form understood by the agents of the legacy MAS. Since the ontology of a service is service-dependent, this translation is impossible to be made by the gateway agent itself. For this purpose the developer must create suitable software that will enable the translation process, this is referred to as a translation module. Consequently, any message that is passed via the *sendRequest*[1] method or extracted from the content of an ACL message (holding the results of a requested service) has to be first parsed by the translation module so as to be understood by either of the agents i.e. the external FIPA agent or the internal agent of the legacy MAS. For instance, a request on an external service is performed via the gateway agent's *sendRequest* method, which receives as input-parameters the name of a service (*ex_service_name*) and a message (*message*) with content being the parameters associated with the service. These parameters are application-dependent; an example in given in given in *section 5.5 (Code 5.4)*. The content of the message sent to the external FIPA agent (providing the service indicated by *ex_service_name*), has to be first translated by a translation module discussed above.

5.3.2. Enabling a legacy MAS to be FIPA interoperable

After the creation of the FIPA-compliant gateways and the configuration of the corresponding gateway agents, the procedure of enabling FIPA interoperability of a legacy MAS involves two simple steps, which have to be executed in order: (a) the initialisation of the FIPA agent platforms that realise the gateways, and (b) the initialisation of the corresponding gateway agent(s). The initiation of the gateways is achieved by the execution of the *startFIPAOS* script from a console, which:

i. initialises the Naming Services e.g. RMI/CORBA used by the platform, which provide the mechanism for agents on a platform to locate one another using a name resolution service i.e. a mapping between the name of an agent (or entity such as the ACC) and its physical location, such that other agents (or entities) can interact with them.

[1] *The syntax of every method defined in GA API is in* Appendix A3

ii. activates the FIPA-OS AgentLoader that loads/starts-up the AMS and DF FIPA agents which support agent management. AgentLoader also supports additional functionality, which enables agents to be managed (shutdown or started) dynamically via a GUI by a user as required.

iii. starts the ACC, the gateway to remote platforms required to interact with other agent platforms.

5.4. Advantages and limitations of the FIPA-compliant gateways

The proposed approach of using the FIPA-compliant gateways for conforming a legacy MAS into a FIPA-compliant one, yields the following advantages:

- *Automatic FIPA interoperability with no or limited knowledge of FIPA specifications.* The adoption of the FIPA-compliant gateways automatically enables a legacy MAS to be FIPA compliant, capable of interoperating with any FIPA-compliant system. The inheritance of FIPA compliance by a legacy MAS involves the creation of the gateways and the configuration of the corresponding gateway agent(s). As described in *section 5.3*, the creation of a gateway is achieved by the execution of a simple script, where the configuration of a gateway agent involves a few lines of code. Therefore, a developer does not have to have any knowledge of the FIPA specifications for conforming a legacy MAS to a FIPA-compliant one; consequently, saving time in terms of reading, understanding, applying the FIPA specifications to a MAS that needs to address FIPA compliance and testing its interoperability. Limited knowledge of FIPA specifications will be required for extending the default gateway agent to support performatives currently not provided by the GatewayAgent API. This concerns knowledge about ACL message structure and the performative specifications that needs to be supported by the default gateway agent, as specified by FIPA.

- *System's architecture remains the same as before.* Implementation is only needed for the gateway agent(s) and their interaction with the internal agents of the system that require or provide a service. The gateways introduce FIPA compliance to a legacy MAS without influencing its original architecture. The interoperability part of the architecture (i.e. the gateways) are *isolated* from the rest of the architecture. Based on FIPA, developers should conform to the latest specifications to guarantee a 100% FIPA-compliant system. Since 2002, FIPA approved the promotion of 23 experimental specifications to standard status[58]. Nowadays, 24 specifications are in standard status and 14 in experimental status.

The advantage of *isolating* the gateways from the rest of a system implies firstly that the original architecture of the corresponding legacy MAS is kept intact and secondly that any new standards (the FIPA revised specifications) which may be released in the future could be covered by a newer release of the GatewayAgent API.

- *Security is increased.* Specifications pertaining to security within the context of the FIPA specifications were started at the beginning of 1997, with the FIPA97 agent management specification[63] and the FIPA98 agent management security specification[64]. There are still no coherent agent security details from FIPA at this time. In fact both of these specifications have now been declared obsolete by FIPA; the management specification has been superseded by new specification but which contains no reference to security. Nevertheless, FIPA is planning in the future to investigate security related issues within FIPA architecture and formulate a long term strategy for the integration of security features into FIPA specifications[23][62]. There is currently debate as to whether a generic or default level of agent security ought to be specified. It is also required that such security criteria should be applicable to different types of agent infrastructures and application domains[119] since:

  - security is a complex issue in the context of MAS and generally system level security,

  - security is part of the software infrastructure in which the agent platform is embedded and is outside the scope of an agent architecture,

  - security is domain and platform (implementation) specific, there is no general agent security architecture which is suitable for all applications and implementations,

  - the focus has been on the development of collaborative, rational agent services within Intranets. Some agent systems do not need security.

Based on the proposed approach, *isolating* the interoperable part of the architecture (i.e. the gateways) from the rest of the system increases security. The policy of the architecture remains hidden to a foreign Agency due to the FIPA-compliant gateways which act as a shield for the core system. The interaction

between the system and a foreign agency is managed by the gateway agent; the rest of the agents, hardware/software resources cannot be accessed. Securing the FIPA-compliant gateways, from where foreign malicious agents can enter into the system, implies minimum security for the rest of the system. Imagine that agent authentication handled by the gateway agent for agent conversations/migrations could work as a firewall for the legacy MAS to restrict access to agents (instead of ports, as a traditional firewall does). The more secure the FIPA-compliant gateways are, the less security is needed for the rest of the system. For instance, the cost of encrypting the messages transmitted between the agents, apart from the gateway agent, can be avoided. Consequently, the minimisation of security (apart from the FIPA-compliant gateways) also increases the overall performance of the system. The gain in performance in a system is therefore related with the number of its internal agents and the message flow between them. Requirements and design issues for adding security to FIPA agent systems can be found in [119].

The gateways are limited in scope - as not all FIPA performatives are supported. Only seven performatives (out of 22 in total) are supported, see *section 5.3.1.1.2*. These seven performatives have been chosen because they can provide interoperable communication between agents hosted on different types of platforms in the context of handling a request or a cancellation of a prior request.

However, if more complex interaction is necessary, such as negotiation, co-operation or co-ordination of heterogeneous agents, the gateways must be extended. For instance, negotiation between heterogeneous agents is handled based on a group of performatives that define how a proposal may be expressed, the preconditions that have to be satisfied for a proposal to be accepted, agent acknowledgement of whether a proposition is true or false. The performatives that belong to this group are: CALL FOR PROPOSAL, PROPOSE, ACCEPT/REJECT PROPOSAL, QUERY/INFORM IF, CONFIRM/DISCONFIRM, REQUEST WHEN/WHENEVER.

The gateway architecture supports such extension - and this can be achieved by defining the required performatives. A Java Class template to define a performative not supported by the default GatewayAgent API is in *section 5.3.1.1.2 (Code 5.2)*.

5.5. Introducing interoperability in SARA architecture using FIPA-compliant gateways

The introduction of FIPA interoperability into the SARA system enables it to communicate with other FIPA-compliant MAS and vice-versa. The union of the SARA system with other MAS(s) extends its capabilities by providing users with further services/information resulting in enhanced data fusion. For instance, information retrieved from the SARA system can be further enhanced by additional information gathered from a GIS (Geographic Information System) that is capable of interoperating with SARA. The longitude and latitude of a particular area of the earth can be used as parameters on a GIS to retrieve land information such as street names, which can then be combined with SARA image(s) of the corresponding geographical coordinates, resulting in a detailed map of the particular area. Likewise, an external FIPA-compliant MAS can interoperate with SARA and use its information resources.

The interoperability of the SARA system is based on the adoption of the FIPA-compliant gateways which are implemented using the GatewayAgent API. The architecture of the SARA system with added FIPA interoperability is depicted in Figure 4.1 of previous chapter. An external multi-agent system (EX MAS) can interoperate with SARA through the FIPA-compliant gateway (outlined by the dashed box) which is placed on every web-server, where SARA can interoperate with an EX MAS through the FIPA-compliant gateway which is placed on every information-server. A detailed representation of the SARA FIPA-compliant gateways' architecture is depicted in Figure 5.7, which is a slight variation of the architecture of FIPA-OS *configuration case 2*[61]; the FIPA-OS toolkit has been used as the FIPA agent platform for the realisation of the FIPA-compliant gateways.

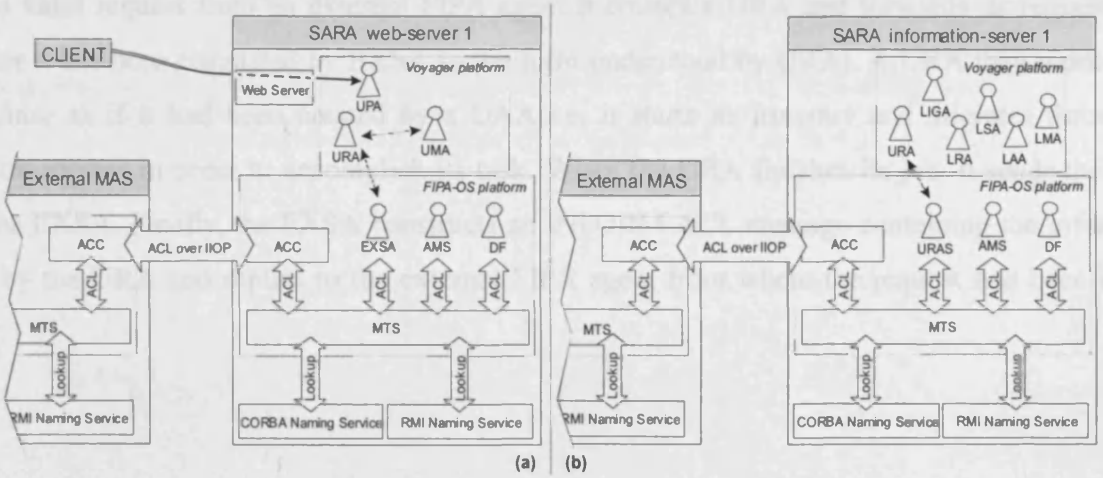


Figure 5.7. Representation of the FIPA-compliant gateways:
(a) on a web-server and (b) on an information-server

Figure 5.7 highlights the logical architecture of the SARA FIPA-compliant gateways, and the flow of information between these and an external MAS. At the bottom of each diagram the available Naming Services (CORBA and RMI) provide the mechanism for agents on a platform to locate one another using a name resolution service. In order for an agent to be located on a platform, it must register with at least one naming service that is used by the platform. In the FIPA-compliant gateways, the registration of AMS and DF to the naming services is handled by the FIPA-OS toolkit, whereas for the gateway agents (EXSA and URAS) this is done automatically by implementing the GatewayAgent interface, see code-example in Code 5.1 (line 26). The Message Transport Service (MTS) on each platform provides a message routing service to enable local and remote agents to exchange ACL messages between themselves using different types of Message Transport Protocols (MTP). Local agents communicate between themselves using the RMI MTP, whereas remote ones use the IIOP MTP. ACC provides the ability for agents on a platform to interact with agents on other platforms. In this instance, a message from a SARA gateway agent to an external FIPA-compliant agent is routed via the platform's ACC. The ACC will then lookup the ACC of the remote platform using its internal database[2] of ACC details. Upon receipt, the receiving ACC passes the message to the external FIPA-compliant agent.

The EXSA (EXternal Service Agent) agent is the gateway agent of the FIPA-compliant gateway placed on every web-server. The service provided by the EXSA is the retrieval of a collection of Earth images from the SARA DL based on specific coordinates. EXSA can be considered similar to UAA. As a user is represented by a UAA, an external FIPA-compliant MAS is represented by an EXSA. Once an EXSA receives a valid request from an external FIPA agent it creates a URA and forwards its request to the latter (after it has been translated by EXSA to the form understood by URA). A URA then works in the same manner as if it had been created by a UAA i.e. it starts its itinerary and migrates through the information-servers in order to accomplish its task. When the URA finishes its job, it sends the results back to the EXSA. Finally, the EXSA constructs an INFORM ACL message containing the information gathered by the URA and replies to the external FIPA agent from where the request had been initially placed.

[2] *The profile of an ACC contains details on the available external agent platforms, see example in* Appendix A2 (Code A2).

The ontology of the service provided by EXSA is depicted below. As mentioned in *Chapter 4 - section 4.3* a DTD is used as a simple instance of an ontology.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT coordinates EMPTY >
<!ATTLIST coordinates c1 NMTOKEN #REQUIRED >
<!ATTLIST coordinates c2 NMTOKEN #REQUIRED >
 ...
<!ATTLIST coordinates c7 NMTOKEN #REQUIRED >
<!ATTLIST coordinates c8 NMTOKEN #REQUIRED >
<!ELEMENT ex_SARA_mes ( coordinates ) >
```

Code 5.3. EXSA's service ontology *(EX_SARA_ontology.dtd)*

An example of an ACL message sent to the EXSA gateway agent by an external FIPA agent requesting a collection of SARA images of specific coordinates may be:

```
(request
 :sender agent_from_EX MAS_id
 :receiver EXSA_id
 :content (<?xml version="1.0" ?>
          <ex_SARA_mes>
          <coordinates c1="33.132"
                       c2="-115.196"
                       c3="33.501"
                       c4="-114.607"
                       c5="32.775"
                       c6="-113.969"
                       c7="32.409"
                       c8="-114.555"/>
          </ex_SARA_mes>)
 :language XML
 :ontology EX_SARA_ontology.dtd
 ...
)
```

Code 5.4. Example of an ACL message received by EXSA

The message that is generated after the translation by EXSA into the form understood by its internal agent i.e. URA, based on the SARA ontology described in *Chapter 4 - section 4.3* would be:

```
<?xml version="1.0" ?>
<!DOCTYPE message SYSTEM "message.dtd">
<Message type="request" id="CLIENTID">
  <Context sender="//web_server1/EXSA_id"
           receiver="//web_server1/URA_id"
           returnby="11/06/04 5pm" />
  <Content>
  ...
    <querydef>
      &trackquery;
    </querydef>
  ...
  </Content>
</Message>
```

Code 5.5. Message sent from EXSA to URA

where *trackquery* contains the task required by URA to be accomplished according to the coordinates specified in the content of the ACL message sent by the external FIPA agent. An example of the *trackquery* is represented in Code 4.3 of the previous chapter, in *section 4.3*. The ontology used by the EXAS's service to transform a request (of a collection of SARA images) into XML format is a simplification of the ontology used by the SARA agents i.e. UAA for the same purpose, depicted in Code 4.4 of the previous chapter in *section 4.3*. The reason for simplification is to make the EXSA service's ontology as easy as possible for developers that need to interoperate with SARA to comprehend. The conversion of an external agent's request to the form understood by the URA internal agent of SARA system is carried out by EXSA.

Finally, the URAS (URA Servant) agent is the gateway agent of the FIPA-compliant gateway placed on every information-server. The purpose of this agent is to serve URA(s) with information gathered from external FIPA-compliant MAS(s). When URA needs to access an EX MAS, its request has to be first translated to the form understood by the external FIPA agent based on the service ontology, before it is send to URAS. Once URAS receives a request from a URA it comes in contact with the appropriate external FIPA agent to accomplish URA's request. By the time URAS has not acquired the results requested by URA, URA is free to continue with its next task (if it has one), migrate to another information-server or wait for URAS agent's response; therefore it uses an asynchronous update model. The list of the external FIPA-compliant multi-agent systems that SARA can interoperate with is controlled by the SARA management agents. Therefore, URA has the right to come in contact with URAS if and only if one or more external MAS(s) is listed in its itinerary.

5.6. Conclusion

This chapter has described how a developer may adopt the generic FIPA-compliant gateways approach for achieving automated FIPA compliance to a legacy MAS. The advantages and limitations of the proposed approach as well as the steps of deployment and how it is possible to extend the generic FIPA-compliant gateways to support other performatives not defined by the default GA API have also been discussed. Experimental tests conducted on the SARA MAS (which adopts the FIPA-compliant gateways for inheriting FIPA compliance) demonstrate the successful interoperability provided by the proposed approach and can be found in *Chapter 8*.

Chapter 6. Load balance in SARA architecture

6.1. Introduction

Load balancing (LB) is one of the most important techniques that can be applied to support the management of agents within a MAS, because apart from achieving the even distribution of agent load among the servers, the management agents' information on LB may also be reused to extend the scalability of a MAS. This chapter presents the dynamic LB mechanism of SARA architecture which is based on a combination of the state-based and model-based approaches of LB. Although the techniques applied to a MAS must be *tailored* to its needs and functional purposes, the model of the proposed technique on LB may be easily amended to support other active archival systems.

6.2. Choosing the appropriate LB technique for SARA

In *Chapter 3 - section 3.3.3* different approaches on LB have been discussed. Research experiments[95] prove that dynamic LB outperforms the static placement scheme by 30-40%, consequently the focus on choosing an efficient load balance technique for SARA is based around dynamic LB.

The objective of market-based approaches on LB is to value resources and achieve an efficient match of supply and demand for resources. This may be achieved by using only a price, match offers and bids, or by employing more sophisticated auction protocols. Therefore LB in this case is directly related and influenced by the amount of currency the agents have. The higher the currency possessed by an agent, the more advantageous it becomes in utilising server resources. Even in the vickrey auction (in which the price paid by the winner agent of the auction equals the second-highest bid placed), agents with less currency have limited chances of winning an auction i.e. utilising resources for the execution of their tasks. Consequently, market-based approaches tend to be priority-based.

The aim of LB in the SARA multi-agent system is to evenly distribute agents among the servers, as well as to equitably serve them. The agents' tasks are carried out simultaneously and there are no priorities between the agents; agent task completion times therefore do not necessary imply a higher priority. Since the objective in SARA DL is to serve equitably the agents without any priority levels, market-based approaches are impractical for this architecture.

As has been mentioned in Chapter 3, the state-based approach of LB is based on information about the system state, which is used to decide the server where a task must be started. Consequently, the nature of information acquired impacts the effectiveness of the LB technique. In addition, in distributed systems where network and server conditions change dynamically, for LB to be effective, it should adapt quickly to those changes. LB approaches which use mobile agents to roam through the network searching for free available resources, lack this kind of adaptiveness. In such approaches agents have to migrate from server to server until they find the needed resources, which is likely to result in network load and in the increase of servers' utilisation. This is because multiple agents simultaneously migrate thought the network and since they are active they consume resources e.g. memory. These agents only have information regarding the servers they have visited, but during their itinerary a lot of changes might take place on the previously visited servers of which the agents will be unaware. For instance, during an agent's itinerary, resources on a server that has already been visited might become available, but the agent will keep on migrating because it is impossible for it to be informed about this change.

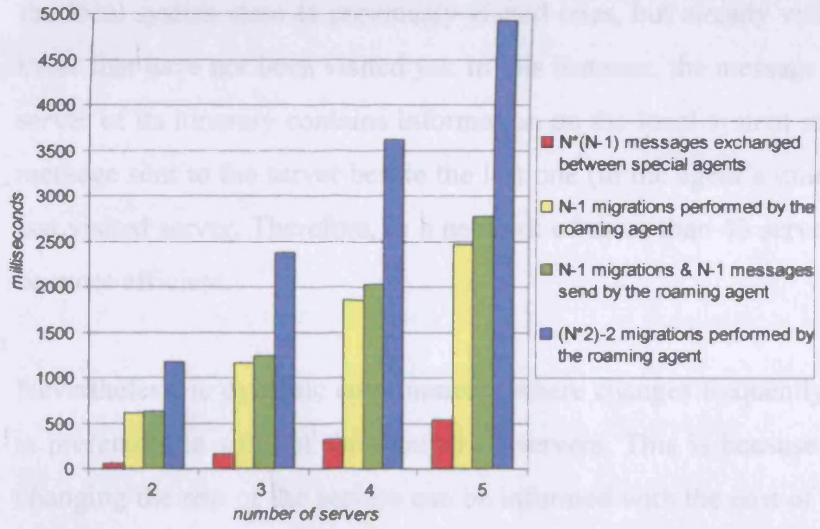
The following three sub-sections discuss the usage of a special agent positioned in every server for gathering, distributing and updating the system state information along with the advantage of having control over load balancing decisions to distribute the agent load among the available servers in a network.

6.2.1. Gathering, distributing and updating system state information

The ability of an agent to have knowledge of the overall system state can be achieved with the introduction of special agents on every server which monitor the local server resources, and exchange their information between themselves. This overcomes the adaptiveness problem, optimises the load balance decisions based on the overall system state information and decreases the network load by eliminating unnecessary agent migrations. A message is faster to transmit in contrast with the time an agent needs to be serialised and migrated. Therefore, where there is a sole roaming agent to gather the overall system state in a network of N servers, such an agent has to be serialised and do N-1 migrations, while the special agents have to exchange $N*(N-1)$ messages between themselves to achieve the same result. However, the roaming agent after it has finished its itinerary, has to either migrate back to all the previously visited servers (i.e. do additional N-1 migrations) or send each of them a message (i.e. total of

N-1 messages have to be transmitted) containing information on the local system states that have been collected so that every server is aware of the overall system state.

An experiment on a 100Mbit/s Fast Ethernet network of five servers was conducted to compare the approach of using a roaming agent in contrast with the existence of special agents on each server for gathering and distributing the overall system state information between the servers. The servers used were Intel Pentium 4 of similar CPU processing powers ranging from 1.8 to 2.1 GHz running Microsoft Windows XP utilising the Voyager[154] agent platform. The experiment was conducted on unloaded servers with virtual local system state information ranging from 150 to 200 bytes each, consisting of information about the server's utilisation, number of virtually active agents, and availability of resources. The initial size of the roaming agent was 2.8Kbytes with the functionality of migration and storing local system state information within. Note that the size of the roaming agent was increasing on each migration due to update in state information that had to be kept. The time needed for a single message (containing local system state information of a server) to be transmitted was 21-36ms, the time of agent serialisation was 31-47ms, whereas the time required by the agent to migrate itself and store the local server system state information was 564-678ms. The time needed to create a reference to a proxy (i.e. special agent) was 93-125ms, but it has not been considered in the evaluation since it is only needed once during the lifetime of the special agents.



Parameter	*Value*
Local system state information	150-200 bytes
Size of roaming agent	2.8 Kbytes
Message transmission	21-36 ms
Agent serialisation	31-47 ms
Agent migration	564-678 ms
Creation of reference to a proxy	93-125 ms

Figure 6.1. Comparison of roaming versus special agent

The number of servers in the chart of Figure 6.1 starts from 2, since a remote agent migration requires the existence of at least two servers. The red and yellow data series represents the time needed from both approaches to gather the initial overall system state, where the green and blue includes the additional time (of the roaming agent approach) to distribute information to all the servers. As can be seen the special agents approach outperforms the roaming agent one.

However, in order to observe the behavior of both approaches in a network of tens of servers, because it was quite difficult to run the same experiment for more than five servers (due to lack of availability in computing facilities), based on the information provided in the table of Figure 6.1 it has been calculated that for N equals to 44, the time required in the special agents approach to exchange $N \cdot (N-1)$ messages is greater than N–1 migrations and N–1 message exchange (performed in the roaming agent approach). This is due to the fact that every message exchanged between the special agents containing local system state information of a server is approximately the same in size, whereas in the roaming agent approach each message sent to a server differs in size. Although the time required in the special agents approach to transmit $N \cdot (N-1)$ messages corresponds to the same number of local system state information that have to be exchanged, in the roaming agent approach the time required to send N–1 messages corresponds to the transmission of $N \cdot (N-1)/2$ local system state information. This is because during the roaming agent's migration, every visiting server is informed by the agent concerning the local system state of previously visited ones, but already visited servers do not have information on those that have not been visited yet. In this instance, the message sent from the roaming agent to the first server of its itinerary contains information on the local system state of every visited server, whereas the message sent to the server before the last one (in the agent's itinerary) contains information only for the last visited server. Therefore, in a network of more than 43 servers roaming agent approach turns out to be more efficient.

Nevertheless, in dynamic environments where changes frequently take place the special agents approach is preferable in spite of the number of servers. This is because in the event of a server system status changing the rest of the servers can be informed with the cost of exchanging N-1 messages, while in the alternative approach, the roaming agent has to perform its task from the beginning i.e. in the best case, do N-1 migrations and exchange N-1 messages (of greater size in comparison with those exchanged in

the special agents approach). Of course, the ideal approach in a network comprised of more than 43 servers, would be to initially gather the system state information using a roaming agent but keeping it updated with the use of special agents placed on each server.

6.2.2. Special agents in the state-based load balance

Although the approach of using special agents in contrast with roaming agents for gathering, disseminating and updating the overall system information is preferable, different policies exists in relation to the special agents' perspective of the system. Policies range from Direct-Neighbor policy (i.e. every special agent communicates only with its direct-neighbor special agents and exchanges local system state information only with them; and LB actions are limited to two direct-neighbor servers) to All-Neighbor policy, where all special agents exchange local system state information between themselves.

Research experiments show that policies where the special agents' perspective of the system is limited suit well highly dynamic applications. In "slowly dynamic"[1] applications[36], the wider the special agents' perspective is, the better the load balance quality that can be achieved; and the total number of migrations can also be diminished. Zambonelli[169] introduced a new scheme of information exchange in neighboring load balance policies, in which the system state (load) information transmitted is distorted to enable special agents to take into account a wider perspective of the system and overcome the limit of the local view. This is achieved by weighting the load of a server with the average load of its neighbour servers. However, his experiments show that the transmission of distorted load information provides high efficiency unless the dynamicity of the load becomes too high i.e. near to 70%, in which case it is preferable to exploit non-distorted load information.

A different approach to achieving efficient LB in neighboring load balance policies with respect to the global view of a system was followed by Keren and Barak[95], with the ability of the special agent to dynamically change their neighbors i.e. their perspective. In their framework for parallel computing, each server's utilisation (load index), which is the only system state information shared between the special agents, is exchanged using two simultaneous dissemination schemes. First, each server -

[1] *According to [36] these are systems that do not change frequently.*

represented by a special agent - sends its load index by attaching it to messages sent by its local agents to other servers. Load indices are also sent to randomly chosen servers using a probabilistic load exchange algorithm. The net result is that for each time unit, every special agent has information about a subset of other special agents. The load balance migration decisions are conducted by the special agents periodically in an asynchronous manner to determine the performance gain in migrating some of its agents to other servers, which is a function of the resulting change in the load and the inter-server communication. If a substantial gain is obtained the migration follows.

Despite the fact that in dynamic environments the narrower the perspective of special agents (i.e. information only about neighboring servers), the better the LB that can be achieved, in the SARA architecture the special agents must have a global view of the system. This is because the architecture of SARA is designed for an active digital library, composed of a collection of different information and computation resource servers (though some might be replicated). Since the agents' tasks are resource-dependent and the resources needed by each task are unknown before its initiation, efficient LB can only be achieved with a global view of the system provided by special agents.

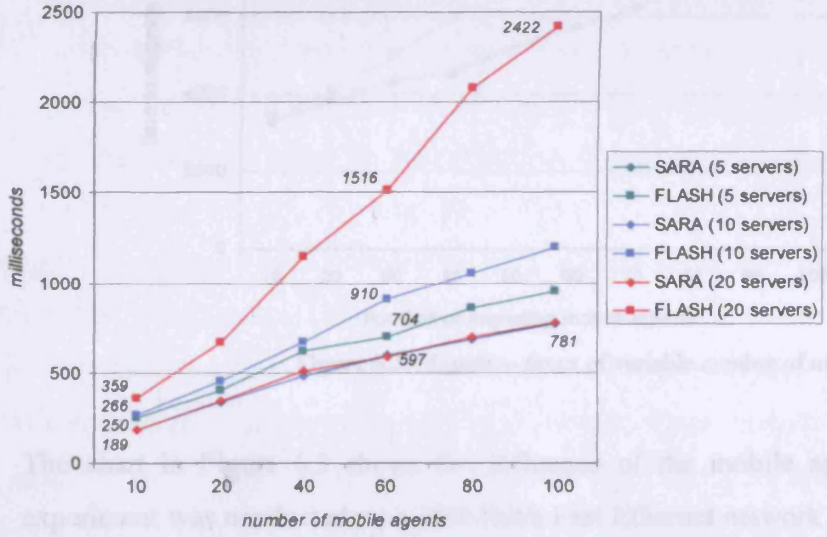
6.2.3. Special agents with control over the LB decisions

The architecture of the FLASH framework (see *Chapter 3 - section 3.3.3.2*) utilises special agents with a global view of the system for gathering, disseminating and updating the system state information. The same idea is followed by the SARA system. The main difference between FLASH and SARA lies in the capability of the special agents. In FLASH, LB decisions are supported by the mobile agents based on their intelligence and the global system state information supplied to them by the special agents. In SARA the control over LB decisions is made by the special agents, referred to as *management* agents. Therefore, in SARA the management agents also optimise the load on the available servers. Although the mobile agents may be programmed with the intelligence to give priority to the overall system optimisation and not on their own tasks, giving management agents the control over the load balance decisions leads to the following benefits:

i) *minimisation of information transmitted*: The management agents balance the load of mobile agents among the servers by defining their itinerary. Once a mobile agent is created, it communicates with

its local management agent, gives its requirements i.e. specifies its task, and waits for a response. The management agent in return, based on the agent's requirements and the current system state information, constructs the mobile agent's itinerary and sends it back to that agent. Consequently, only two messages are exchanged between a mobile agent and a management agent: the agent's requirements and the agent's itinerary. In the case where the mobile agent would be in control of the LB decision, every mobile agent would have to retrieve from a management agent the overall system state information in order to make a reliable decision; which results in unnecessary duplication of information i.e. the same information sent to different agents.

The chart in Figure 6.2 shows the different time spent (in milliseconds) on the interaction of a special/management agent with a number of mobile agents on a single server, according to the amount of data that have to be exchanged based on what has control over the LB decisions. The experiment was conducted on an Intel Pentium 4 1.8Ghz server running the Voyager agent platform on Microsoft Windows XP, where an agent's itinerary needed 15 bytes per server, its request was approximately 60-80 bytes and the system state information of a server encoded in XML at 700-750 bytes. An example of XML schema used to encode the overall system state information can be found in Code 6.1 of *section 6.3.1.3*; the "LOCAL" tag encloses the system state information of a single server.



Parameter	*Value*
Agent itinerary	15 bytes/server
Agent request	60-80 bytes
System state information of a server	700-750 bytes

Figure 6.2. Interaction between the special/management agent and the mobile agents

Although the difference in time between the two approaches is minimal i.e. a few seconds, as the number of servers on the network is increased considering the total time of the agents' interaction from each server, this difference becomes important. Finally, from the above chart it can be observed that in SARA the agents' interaction time, irrespectively of the variable introduction of participants is almost uninfluenced by the number of servers (from 5 to 20) employed in the network.

ii) *minimisation of the mobile agent's size*: Decisions on LB are based a model that accepts as input an agent's requirements and the system state information, and gives as output an itinerary of servers where the particular agent should migrate to. In SARA, the management agents provide this functionality and are stationary. Alternatively, every mobile agent must have this decision support algorithm within itself. One of the most important characteristics of a mobile agent is its size; the smaller the mobile agent is in size, the faster it can move through the network. Hence, by giving the management agents the control over LB decisions, the size of the mobile agents is preserved to its original size i.e. the LB model is not contained within the mobile agent.

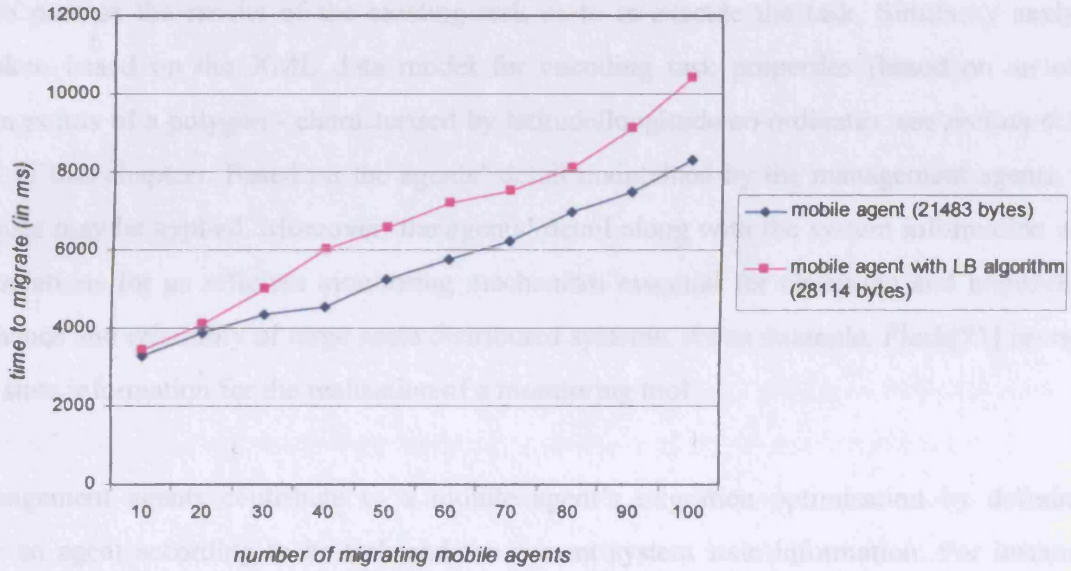


Figure 6.3. Migration times of variable number of migrating agents

The chart in Figure 6.3 shows the influence of the mobile agent's size on its migration. The experiment was conducted on a 100Mbit/s Fast Ethernet network with two Sun Ultra 5 Workstation of a 270 MHz UltraSPARC-IIi 64-bit processor running on Solaris 8, utilising the Voyager agent

platform. In the experiment two types of mobile agents with different sizes were used. One of them was the actual URA mobile agent used in SARA of 21483 bytes and the second one was the same agent with the load balance decision algorithm within, resulting in an agent of 28114 bytes (23.5% larger in size than the first one). As can be observed, the total migration time increases with the number of concurrently migrating agents, and the larger a mobile agent is in size the more time is required for its migration when the number of concurrent migrating agents is increased.

iii) *system optimisation*: Management agents also maintain a record about mobile agents that are active on their host platform. This information includes the task of the agent, the resources that have been used, the time of completing the task and the site where the results of the task have been stored. This information is used to support LB, and for undertaking similarity analysis between agent requests. Hence, if an agent's task (request) is identical to a task already performed, the task does not have to be repeated and previous results can be retrieved. If an agent's task is similar but not exactly the same as an already accomplished task, the model determines if it is worthwhile for the agent to process the results of the existing task or to re-execute the task. Similarity analysis is undertaken based on the XML data model for encoding task properties (based on an overlap between points of a polygon - characterised by latitude/longitude co-ordinates, see *section 6.3.2.6 - Case 1* of this chapter). Based on the agents' detail maintained by the management agents, cache techniques may be applied. Moreover, the agents' detail along with the system information can lay the foundations for an efficient monitoring mechanism essential for observing and improving the performance and reliability of large scale distributed systems. As an example, Flash[71] re-uses the system state information for the realisation of a monitoring tool.

Hence, management agents contribute to a mobile agent's migration optimisation by defining the itinerary for an agent according to its task and the current system state information. For instance, an agent with a task of acquiring a collection of images and filtering them on a compute server against a user's custom analysis algorithm will be guided by a management agent as to which server it should migrate to. *How* the agent migrates (i.e. just the agent itself, the agent containing the custom algorithm etc.) and *when* it should load any classes necessary for the accomplishment of its task, is its own choice based on its intelligence and the status of its task. Even in the event of a server failure, the mobile agent

is capable of moving autonomously to the next available server of its itinerary, communicating with the local stationary agents without being controlled by the management agents.

6.3 The SARA LB mechanism

The SARA LB mechanism is an approach towards the combination of the *model-based* and *state-based* approach with the objective of minimising the mean flow time, maximising resource utilisation and minimising the mean response ratio[146]. Mean flow time is the average time from when a task is created to when it is completed. It is assumed that resources spend most of their time serving agents. Hence, resource utilisation is the percentage of time a resource is used by agents to undertake their tasks. Mean response ratio is the average ratio of the actual time to complete a job divided by the time to run that task on an unloaded benchmarked server.

The agents' tasks are classified into simple and complex analogous to their nature. Simple tasks are characterised as those that are related with the data gathering procedure whereas the complex ones are those that filter the data retrieved from a simple task e.g. based on an image processing algorithm, requiring more processing power and time, and they are more rare. Furthermore, the agents' tasks are preemptive i.e. running processes may be suspended, moved to a remote server and restarted. For instance, an agent after it has collected the appropriate data from a database/archive server can proceed with their filtering by migrating to a compute server.

The architecture of SARA LB utilises special agents, referred to as *management agents,* with a global view of the system for gathering, disseminating and updating the system state information. Decisions on LB are supported through the management agents. As in most of the systems that explore the model-based approach to LB, use distributions of CPU load and expected process lifetime to decide if and when to migrate, similarly the SARA model is mainly based on the servers' utilisations where emphasis is given on prediction of the complex agent's task lifetime. The model adapts over time due to the information gathered from the *state-based* approach. The SARA LB mechanism is generic, and can be easily adapted for other MAS, especially those designed for agent-based Digital Libraries that provides computing services in addition to data-retrieval services; so that users can initiate computing jobs on remote supercomputers for processing, mining, and filtering of the data in the Library.

Sections 6.3.1 and 6.3.2 describe in detail the state-based and model-based part of the SARA load balance. Discussion on the adaptability of the SARA model can be found in section 6.4 of this chapter.

6.3.1. State-based LB in SARA

This section describes the *state-based* approach part of LB, which is responsible for gathering the information used by the model (of the *model-based* approach), and presents the architecture needed to support LB with reference to the SARA MAS. This involves the roles and the position of the management agents in the network, the interaction between themselves and the other agents of the system, the acquisition and distribution of information among the management agents.

6.3.1.1. The management agents in the SARA architecture

The SARA system is composed of a collection of information-servers and web-servers, each of them having a group of agents. A mobile agent (URA) is assigned to every user that needs to access the system. As the number of users increases so do the mobile agents. Essentially, the performance of the SARA system is based on the rapid and successful accomplishment of the mobile agents' tasks. A management agent exists for every server. Every information-server has a LMA (Local Management Agent), where every web-server has a UMA (Universal Management Agent). Although the LMA and UMA management agents differ in their capabilities and the kind of information they posses, their common objective is to optimise system performance.

Having a management agent on every web-server from where URA agents are initially created, extends the possibility of optimising the agents' itinerary, their tasks and therefore the overall performance of the system. Once a URA migrates to an information-server, its itinerary can then be managed by the corresponding server's LMA. Unnecessary agent moves e.g. migration to servers with unavailable resources, results in the delay of the agent's task and contributes to the increase of network traffic. Hence, identification of an agent's request and a comparison with a previous query is preferable before the initialisation of an agent's itinerary. In the event of similar requests the UMA will decide if it is worthwhile for the agent to process the already stored results in order to accomplish its task. The functionality of UMA to work on past queries is similar to a query caching technique.

Every LMA maintains information about its local server's state and of other servers it interacts with. An LMA informs a visiting agent of any changes that take place in the network, if these concern the visiting agent's task. Similar to the UMA, the LMA is responsible for optimising the itinerary of any visiting mobile agent with respect to balancing load and the agent's requirements i.e. to ensure that the next server the agent visits is of relevance with respect to data acquired at the current site.

A benefit of multiple management agents over a centralised scheme is that the system does not have a central point of failure. If there is a failure in one of the management agents, the system can operate with all the remaining ones. Moreover, in the centralised scheme as the number of agents increase, the network load is increased dramatically due to the fact that all agents have to report to and be managed by a single management agent.

6.3.1.2. Distribution of information among the management agents

In contrast with the centralised scheme where a global database is used to hold all the information for each server, in the distributed scheme the information has to be distributed. The way in which information is distributed among the management agents differs in terms of scope, size and the level of detail provided[167]. In this instance, every management agent of each server has a *map* providing this information.

A map is referred to as *global network map* if it provides all information for every available server in the network. If the information provided in a map is reduced for servers which are not in the local region, the map is referred to as *map of the surrounding area*, whereas a map of information on the local server and the neighbor servers only is referred to as *neighbor map*. There are variations in describing which server should be the neighbor to the local one, and it is up to the developer to decide which one should be adopted e.g. all servers within a sub-network and between sub-networks, two defined servers as being adjacent, or all servers reachable within a certain time are adjacent.

In the centralised scheme, every agent has to connect to the central server in order to retrieve information (by the sole management agent) and every server has to register its details to the central server. Since the total information is stored in one single location the network overload increases dramatically and in the

case of a failure on the central server, the system paralyses completely. In a distributed approach, a map of the surrounding area or a neighbor map imposes agents to have intelligence in order to move in a good enough manner through the network, due to the fact that information in reduced on each server. The agent's intelligence in this context refers to the agent's ability for predicting system state information. The narrower the perspective of the management agents, the more intelligence is required by the agents; how much intelligence an agent needs depends on the level of information available to it.

As discussed in *section 6.3.1.1*, having a management agent on every server of the network extends the ability for optimising the system in various ways. The information maintained by the management agents plays a significant role and the use of a global network map seems to be the most promising solution. In this approach, the amount of information maintained by each management agent is the same as in the centralised scheme. The difference in conjunction with the centralised scheme is that network traffic is reduced since information is distributed and server (or management agent) failures do not affect the whole system. On the other hand, in a distributed approach with the use of a global network map all of the management agents need to exchange messages between themselves in order to preserve the integrity of information held by each of them. This is the reason why this approach may cause network overload if the information that has to be exchanged between the management agents is too much. Network overload is a major factor that influences the performance of a distributed system. The way in which the information is distributed among the management agents i.e. what kind, how much information each management agent should have and how frequent the information is exchanged, has a direct affect on it.

Due to their physical location in the network (on a web-server or information-server), UMA and LMA management agents have different responsibilities and capabilities. Consequently they do not need the same amount of information. The local information of every LMA on its server comprises the system state information and must be maintained both by LMAs and UMAs; whereas information regarding mobile agents' personal details (necessary for identification of similar agent tasks, cache techniques etc.) is needed only by UMAs. Therefore, every management agent uses a global network map to maintain the system state information, while UMAs have additional information on mobile agents' details.

6.3.1.3. Information maintained by management agents

The information maintained by the management agents can be found in Tables 6.1 and 6.2. The left column of each table contains the information held by every management agent, whereas the right column identifies the source of information.

LMA's information is basically divided in two sections. The first section represented by the "LOCAL" label contains information for the local server of the particular LMA, whereas the second one represented by the "REMOTE" label contains information regarding the rest of the available servers in the network. The system state information is composed of the local and remote information on servers. This information is also maintained by every UMA, identified by the grey colour in Table 6.2. In addition, UMAs maintain information on the URA mobile agents. Details regarding local URAs i.e. those that have been launched from the web-server where the particular UMA resides are kept under the "LOCAL" label, whereas details about the URA agents launched from other web-servers are held under the "REMOTE" label. As can be observed from the content of Tables 6.1 and 6.2 highlighted in grey color, the information of LMAs is a sub-set of UMA management agents' information.

Furthermore, every management agent holds information on the network connection bandwidths of the available servers in the network and their local URA agents that become persistent due to unavailability of resources. The information on persistent agents is useful in the event where the server(s) that a particular URA needed to visit which was previously unavailable has become available.

An example of the management agents' information encoded in XML form as exchanged in the SARA prototype can be found in Code 6.1 and 6.2 after the tables. The management agents' information may be further extended by providing a list of foreign FIPA-compliant agent systems along with appropriate connection details with which SARA can interoperate.

Table 6.1. LMA's information

LMA's information	acquired by
MA's SPACE: *server, ID name*	MA creator of SPACE
LOCAL: resources: software: *status of voyager server, available analysis algorithms* hardware: database/archive server: *status, processing power*	local LAA
agents' average completion task time, server's utilisation	LMA itself
compute server: *status, processing power, average data filtered per sec., maximum data filtered per sec.*	local LAA
number of agents: *active, **persistent***	LMA itself
REMOTE: Servers' resources, number of agents:...	LMAs
SERVERS' BANDWIDTHS: server x with server y: *bytes/sec*	LMAs/UMAs
UNAVAILABLE SERVERS: database/archive servers: server x: *agent ID1, agent ID2, agent ID3* compute servers: server y: *agent ID4*	local URAs *(persistent agents)*

Table 6.2. UMA's information

UMA's information	acquired by
MA's SPACE: *server, ID name*	MA creator of SPACE
LOCAL AGENTS' INFO: **agent id**: general: ***request**, time of request*	local UAA/EXSA *(upon URA's creation)*
*time of request accomplished, **status of task*** location of results: ***server's IP**, physical location path, file-space acquired* resources used: software: *analysis algorithm (AA) used, size of custom AA* hardware: *database/file archives used, engagement time (from-to), server's utilisation (before-after), compute server used, engagement time (from-to)*	URA *(before its termination)*
REMOTE AGENTS' INFO: agent id: *request, server, status of the task*	UMAs
LMAs' INFO: server x, y: ...	LMAs
SERVERS' BANDWIDTHS: server x with server y: *bytes/sec*	LMAs/UMAs
UNAVAILABLE SERVERS: database/archive servers: server x: *agent ID1, agent ID2, agent ID3* compute servers: server y: *agent ID4*	local URAs *(persistent agents)*

```
<?xml version="1.0" ?>
<MAN_AGENT_LB>
  <SPACE SERVER="131.251.47.102:8000" NAME[2]="1087552159707"/>
  <LOCAL>
    <SOFTWARE>
      <VOYAGER_SERVER>online</VOYAGER_SERVER>
      <ANALYSIS_ALG>2</ANALYSIS_ALG>
      <ANALYSIS_ALGS>
        <A_ALG>Edge detection (Mexican hat/Marr) Filter - 13x13 matrix</A_ALG>
        <A_ALG>Edge detection (Laplacian) Filter - 5x5 matrix</A_ALG>
      </ANALYSIS_ALGS>
    </SOFTWARE>
    <HARDWARE>
      <DB_SERVER>
        <STATUS>online</STATUS>
        <PROC_P>8.881231293906588</PROC_P>
        <AV_COMPL_TIME>11259</AV_COMPL_TIME>
        <UTILISATION>0.011259700000000001</UTILISATION>
      </DB_SERVER>
      <COMP_SERVER>
        <STATUS>offline</STATUS>
        <PROC_P>111244</PROC_P>
        <AV_DATA_FIL>55342</AV_DATA_FIL>
        <MAX_DATA_FIL>67883</MAX_DATA_FIL>
      </COMP_SERVER>
    </HARDWARE>
    <AGENTS>
      <ACTIVE>2</ACTIVE>  <PERSISTENT>0</PERSISTENT>
    </AGENTS>
  </LOCAL>
  <REMOTE>
    <SERVERS>
      <SERVER ID="131.251.47.171">
        <SOFTWARE/>
        <HARDWARE/>
        <AGENTS/>
      </SERVER>
      <SERVER ID="131.251.42.9">
        ...
    </SERVERS>
  </REMOTE>
  </BANDWIDTHS>
    <SERVER ID1="131.251.47.102" ID2="131.251.47.171" BYTES="250.0"/>
    <SERVER ID1="131.251.47.102" ID2="131.251.42.9" BYTES="300.0"/>
    ...
  </BANDWIDTHS>
  <UNAVAILABLE_SERVERS>
    <INFO_SERVERS>
      <SERVER ID="131.251.42.115">
        <AGENT ID="URA_1087545213223452"/>
        ...
      </SERVER>
      ...
    </INFO_SERVERS>
    <COMP_SERVERS>
      <SERVER ID="131.251.47.171">
        <AGENT ID="URA_1087545210020012"/>
        ...
      </SERVER>
      ...
    </COMP_SERVERS>
  </UNAVAILABLE_SERVERS>
</MAN_AGENT_LB>
```

Code 6.1. LMA's information encoded in XML

[2] *The name of a SPACE is derived from the time when it is created (expressed in milliseconds); along with the IP address of the server hosting the SPACE a unique ID is formed, for further details see* Chapter 7 - section 7.2.1.3.

```
<?xml version="1.0" ?>
<MAN_AGENT_LB>
  <SPACE SERVER="131.251.47.102:8000" NAME="1087552159707"/>
  <AGENT_INFO>
    <LOCAL>
      <AGENT>
        <AGENT_ID="URA_1087545213314" REQUEST="select * from ..." STATUS="done"
                REQUEST_START_T="34534523" REQUEST_FINISH_T="44534535"/>
        <RESULTS_LOCAT SERVER_IP="131.251.42.9"
                RESULTS_LOC="/home/scmcg/public_html/..." FILE_SP_ACQUIRED="12333"/>
        <RESOURCE_USED>
          <SOFTWARE ANALYSIS_ALG="Sharp Filter - 3x3 matrix" CUSTOM_ALG_SIZE="0"/>
          <HARDWARE DB_FILE_USED="Oracle 8.0.1" ENGAGE_START="34534523"
            ENGAGE_FINISH="43453453" SERVER_UTIL_BEF="0.011259700000000001"
            SERVER_UTIL_AFT="11.073988453000045101"
            COMP_SERVER_USED="131.251.42.9" ENGAGE_START2="43453470"
            ENGAGE_FINISH2="44534535"/>
        </RESOURCE_USED>
      </AGENT>
      <AGENT>
        ...
      </AGENT>
      ...
    </LOCAL>
    <REMOTE>
      <AGENT_ID="URA_1087544817833" REQUEST="select * from..." SERVER="131.251.42.171" STATUS="done"/>
      <AGENT_ID="URA_1087544928562" REQUEST="select * from..." SERVER="131.251.42.171" STATUS="pending"/>
      ...
    </REMOTE>
  </AGENT_INFO>
  <LMAs_INFO>
    <SERVERS>
      <SERVER ID="131.251.42.9">
        <SOFTWARE>
          <VOYAGER_SERVER>online</VOYAGER_SERVER>
          ...
      <SERVER ID/>
      <SERVER ID="131.251.47.216">
        ...
      <SERVER ID/>
    </SERVERS>
  </LMAs_INFO>
  </BANDWIDTHS>
    <SERVER ID1="131.251.47.102" ID2="131.251.47.171" BYTES="250.0"/>
    <SERVER ID1="131.251.47.102" ID2="131.251.42.9" BYTES="300.0"/>
    ...
  </BANDWIDTHS>
  <UNAVAILABLE_SERVERS>
    <INFO_SERVERS>
      <SERVER ID="131.251.42.115">
        <AGENT ID="URA_1087545000001122"/>
        <AGENT ID="URA_1087545000445411"/>
        ...
      </SERVER>
      ...
    </INFO_SERVERS>
    <COMP_SERVERS>
      <SERVER ID="131.251.47.171">
        <AGENT ID="URA_1087545200043999"/>
        ...
      </SERVER>
      ...
    </COMP_SERVERS>
  </UNAVAILABLE_SERVERS>
</MAN_AGENT_LB>
```

Code 6.2. UMA's information encoded in XML

6.3.1.4. Communication between the management agents

The management agents exchange information via direct or multicast messages depending on the number of participants that are involved in the message exchange. When there is only one recipient the message that is exchanged is of type "direct"; whereas a message that involves a group of agents is of type "multicast". Most traditional systems use a single repeater object to replicate a message or event to each object in the target group. This approach is appropriate when the number of objects in the target group is small, but does not scale well when large numbers of objects are involved. Voyager uses scalable architecture for message/event replication called SPACE[156]. A SPACE is a distributed container that can span Virtual Machines (VMs). A sub-SPACE is a container that cannot span VMs. A SPACE is created by linking together one or more sub-SPACEs, and its contents are the union of its linked sub-SPACEs. A message/event sent via a multicast proxy into a sub-SPACE is cloned to each of its neighboring sub-SPACEs before being delivered to every object in the local sub-SPACE, resulting in a rapid, parallel fan-out of the message to every object in the SPACE. As the message propagates, it leaves behind a marker unique to that message that is remembered by the sub-SPACE for a period of five minutes. If a clone of that message re-enters the sub-SPACE, the clone detects the marker and self-destructs. The marker allows the developer to connect sub-SPACEs to form arbitrary topologies without the danger of multiple message delivery. The more interconnected the sub-SPACEs are, the more fault-tolerant they become in the face of individual network failures. Figure 6.4 illustrates sending a message to a sub-SPACE in a SPACE.

In SARA architecture a single SPACE is utilised and every management agent is registered to it. Since the LMA and UMA management agents do not share the same kind of information (see previous section), every management agent is subscribed in SPACE to receive system state information, whereas UMAs are also subscribed to receive information regarding URAs' personal details. Therefore, there are two kinds of subscribers referred to as 'LMA-UMA' and 'UMA'.

The system state information is updated as often as there is a status change on one of the information-servers (database/archive or compute server). In the event of a change, the LMA of the information-server where the change took place sends the updated information to the rest of the management agents i.e. to the 'LMA-UMA' subscribers using the SPACE. Changes of the system state information may be

caused either due to a change of a server's resources status or due to URA agents' actions. The URA agents alter the system state information on their arrival/departure on an information-server i.e. when the number of agents on a server is increased/decreased or during their state conversion from active to persistent and vice-versa. Information concerning the personal details of the URA agents is updated on the creation or termination of a URA agent and is only exchanged between the UMAs i.e information is send only to 'UMA' subscribers.

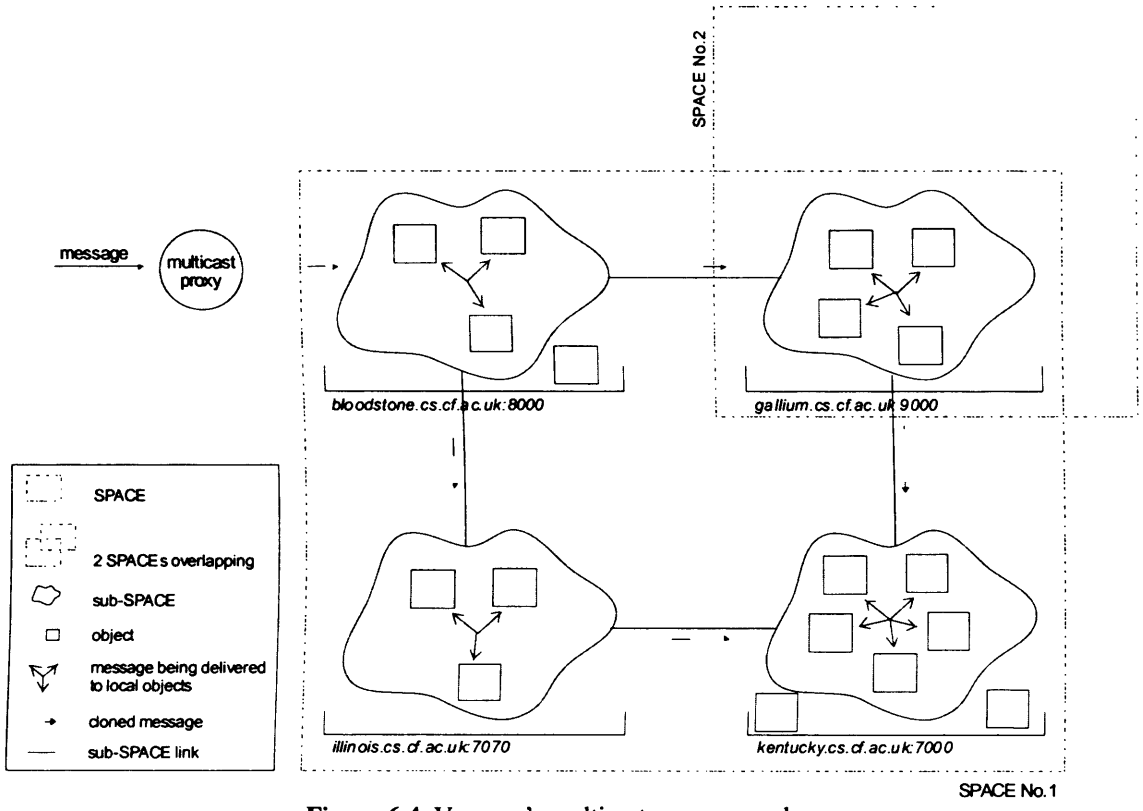


Figure 6.4. Voyager's multicast message exchange

On initialisation, the management agents exchange all of their information between themselves; subsequently only the updated information is exchanged. The management agents' interactions and the kind of information they exchange are based on system events summarised in Table 6.3. The table also identifies the interactions of the management agents with the rest of the agents that update their information. For instance, a UAA serves its local UMA with information related to URAs initialised by the former, where a LAA is responsible of informing its local LMA about the status of its server's resources. However, the status of a server is impossible to be checked by a local agent such as LAA, because agents cannot survive when the Voyager agent platform or the actual server fails. URA has the

capability of receiving callbacks before and after its migration, see *Chapter 7 - section 7.2.2.2.* If its migration fails due to the visiting server's failure, it informs its local management agent i.e LMA or UMA. Afterwards, the particular management agent informs the rest of the management agents by preventing other agents from migrating to the failed server.

A fault-tolerance mechanism is built within the management agents which enables them to automatically recover their state and information, and register properly to the SPACE after a failure. The creation of the SPACE is performed by one of the management agents on initialisation of the system. In the event that the server of the management agent which has created the SPACE fails, the first management agent that sends a message to the SPACE and fails, is capable of re-constructing the SPACE and informing the rest of the available management agents of the new SPACE details[3]. Information on the creation of a SPACE and the interaction of management agents with it are discussed in *Chapter 7 - section 7.2.1.3.*

[3] *The IP address of the server on where the SPACE is created and the SPACE ID name.*

Table 6.3. Management agents' interaction

Event		Interaction (sender – recipient)	Information exchange	Type of message
type	description			
initialisation	creation of MA's SPACE	creator MA - LMAs/UMAs	contents in row 1 of table 6.1 and 6.2	multicast
	LMAs' information exchange	LMA - LMAs/UMAs	contents in grey rows of table 6.1	multicast
	determination of network connection speeds between servers	LMAs/UMAs - LMAs/UMAs	contents in row 6 of table 6.1 and row 5 in table 6.2	multicast
during execution	-//-	-//-	-//-	-//-
	MA that was previously down, tries to determine SPACE	LMA/UMA - one of LMAs/UMAs	contents in row 1 of table 6.1/6.2	direct
	acquisition of URA's itinerary before its task initiation	URA - local UMA/LMA	creation of URA's itinerary based on information in gray rows of tables 6.1/6.2	direct
	upon URA's creation	UAA/EXSA - local UMA	contents in row 2 of table 6.2	direct
		UMA – UMAs	information in bold of table 6.2	multicast
	before URA's death	URA - local UMA	contents in row 3 of table 6.2	direct
		UMA – UMAs	information in bold and underlined, in row 3 of table 6.2	multicast
	arrival/departure of URA to/from an information- server	local LMA – LMAs/UMAs	active agents in row 5, server's utilisation in row 3 of table 6.1 & corresponding information in row 5 of table 6.2	multicast
	departure of URA to/from an information- server	-//-	-//- plus the agents' average completion task time in row 3 of table 6.1 & row 5 of table 6.2	-//-
	URA's state change i.e. from active to persistent and vice-versa	local LMA – LMAs/UMAs	active/persistent agents in row 5 of table 6.1 & corresponding information in row 5 of table 6.2	multicast
	sever will be unavailable until a specified time	LMA – LMAs/UMAs	information concerning the time of when the status of server will change (row 2 of table 6.1)	multicast
	need for further information about an agent's task	UMA – UMA	selected information of row 2,3 of table 6.2 based on the recipient UMA needs	direct
	change on LMA's information *(i.e. status/resources)*	LAA - local LMA	contents in row 2,4 of table 6.1	direct
		LMA – LMAs/UMAs	contents in grey rows of table 6.1	multicast
	change on UMA's information *(concerning URA personal details)*	UMA-UMAs	contents in row 4 of table 6.2	multicast
	upon availability of previously unavailable resources, MA will activate persistent agents	UMA/LMA - URA	activation of persistent URA agents based on information in row 8 of table 6.1 and row 6 of table 6.2	direct
failure	failure on pinging a server to determine network connection latency	LMA/UMA - LMAs/UMAs	information concerning the status of server in row 6 of table 6.1/row 5 of table 6.2	multicast
	failure on sending a message using the SPACE, MA sender tries to find a new SPACE	LMA/UMA - one of LMAs/UMAs	contents in row 1 of table 6.1/6.2	direct
	URA's pre-migration failure	URA - local LMA/UMA	information concerning the status of server in row 6 of table 6.1/row 5 of table 6.2	direct
		LMA/UMA - LMAs/UMAs		multicast
	database connection failure	URA/LRA - local LMA	information concerning the status of database/achive server in row 6 of table 6.1/row 5 of table 6.2	direct
		LMA - LMAs/UMAs		multicast

6.3.2. Model-based LB in SARA

This section describes the model-based part of the SARA LB scheme. Whereas the objective of the state-based part of SARA LB is to gather the system state information, the model-based part has to exploit this information to its optimum level for making accurate decisions to balance the load of agent tasks among the servers. Generally, the models used in state-based approaches are much simpler than those used in model-based ones, since they do not have to predict e.g. the system state, but rather work out the system state information provided by each server. The SARA LB scheme is based on a combination of the system state information and the prediction of agent task lifetimes. Of course, the more reliable the system state information, the more accurate is the outcome of the model. Therefore, the information exchanged between the management agents is a very important factor. Firstly, because the efficiency of the model depends on it (i.e. quality of information) and secondly, the greater the amount of information, the higher the risk for an increase in network load. The main information exchanged in state-based approaches discussed in *Chapter 3*, as well as in SARA LB, is the number of agents on each server and the number of available servers along with their utilisation indices.

A second and most important factor on the task assignment policy in either state-based or model-based approaches of LB [19][22][28][37][48][69][95][98][107][163][166][167] is the utilisation of the servers present in a network, in relation to their processing power. Irrespective of the algorithm each technique is used for the distribution of tasks among the available servers, their common policy is that a task should be assigned to the least loaded server i.e. the one with the lowest utilisation, assuming that the servers are of equal processing power. Consequently, the more accurate the estimation of a server's utilisation, the better the load balance.

6.3.2.1. Estimating server utilisation

The utilisation of a server at any point in time is directly correlated with its load i.e. the tasks which are being executed at that time on the corresponding server. For instance, the utilisation of a server which has to process ten images simultaneously is obvious greater than the utilisation of a server (of identical processing power) which has to process a single image of same properties. Malone[98] has defined that the utilisation of a system can be found by the expected amount of processing requested per time unit divided by the total amount of processing power in the system, given by the following formula:

$$U = \frac{a \cdot \mu}{L}$$

Formula 6.1. Utilisation of a system

where

a = the average number of job arrivals per time unit

μ = the average job length

L = the total processing power in the system

In the SARA model this formula is used to evaluate the utilisation of each server separately rather than the utilisation of the system as a whole. Therefore for a given server, α corresponds to the number of agents on that server (assuming that there is a task per agent), μ to the average task time of the a agents and L to the total processing power of the corresponding server.

Formula 7.1, apart from estimating a server's utilisation in relation to its processing power L (given a collection of servers), also helps determine the server that will be unloaded first i.e. will accomplish all of its tasks sooner than any of the rest of the servers. For example, let us assume that there are two servers S_1 and S_2 of identical processing power, where S_1 has 5 agents with average task time of 50 seconds and S_2 has 10 agents of 20 seconds average task time. The server that is likely - but not 100% guaranteed - to be unloaded first is S_2 and this is based on formula 7.1, as its utilisation will be less than S_1. Accuracy in estimating a server's utilisation is based on perfect estimations of the agent task lifetimes. The more accurate the average task time μ of a agents is, the more reliable the corresponding server's utilisation. Therefore the lifetime prediction/estimation of every agent task is important. The process of making correct estimations on the lifetime of a task is a very difficult procedure and is usually based on the nature of the task. For instance, if a task regards a compilation, the number of lines of code and files to link might be used as a guide. If a task is related to the processing of an image the resolution, size and type of the corresponding image might be considered, whereas the lifetime of a task for transferring files across a network can be estimated by the size of files divided by the available network connection speed.

A simpler approach of acquiring the utilisation of a server is by using specialised routines/utilities (like *xload* or *ps* of Unix operating system) that provide the CPU usage. The ALABAMA[82] model of

FLASH LB scheme is using such routines to acquire the servers' utilisation. The difference between those kind of routines and Malone's approach is that while the former provide the current utilisation (CPU usage) of a server the latter also denotes a value of when a server will be unloaded i.e. its utilisation will be null.

Apart from the fact that a server's CPU usage changes frequently, decisions on load balance which are based on servers' utilisation should not rely on the current utilisation of each server but rather on which server will be unloaded first. Let us assume that there are two servers S_1 and S_2 of identical processing power, where S_1 has to carry out two tasks T_1 of 10 seconds each and S_2 has two tasks T_2 of 20 seconds each; note that both of T_1 and T_2 tasks demand the same amount of processing power, but T_2 needs more time to be accomplished. For the first 10 seconds the CPU usage of both servers will be almost the same, but after the 10^{th} second S_1's utilisation will drop to zero. If the decision as to where a new task should be assigned in the first 10 seconds was based on the current utilisation of each server instead of which server will be unloaded first, there is a chance of 50% that the new task would be assigned to S_2 (since the CPU usage of both servers before the 10^{th} second would be almost identical). In contrast, based on Malone's approach of estimating a server's utilisation it can be inferred that S_1 will be unloaded before S_2, and therefore the new task would be 100% assigned to S_1.

The advantage of using Malone's formula is that apart from estimating a server's utilization, it is also possible to predict its utilisation before the assignment of a new task to that server, given that the lifetime of the corresponding task is known. This could be achieved by adding the time of the new agent task to the product in the numerator in Malone's formula, to predict the server utilisation after that task would have been executed on the particular server. Predictions on server utilisation are performed before the introduction of complex agent tasks to compute servers.

6.3.2.2. Calculation of a server's processing power

The processing power of a server (*L*) in Malone's formula is used to estimate the utilisation of a server. For the calculation of the processing power of a server, a small routine was developed to measure its performance. The routine launches ten URA agents with tasks similar to those executed in real-time in SARA DL and measures the overall time (in milliseconds) needed to accomplish their tasks on each

server separately.

If x is the execution time of the routine, then the utilisation is proportional to x. A lower value of x therefore implies a lower utilisation. Hence, the processing power L of a server is inversely proportional to its utilisation, or:

$$L = \frac{1}{x}$$

Formula 6.2. Processing power of a server

The calculation of a server's throughput on image processing is performed by a similar routine which measures the time needed in milliseconds by each server to process the same image against a specific image processing algorithm. Note that L refers to the performance of the server on providing data repository facilities (for simple agent tasks), whereas its throughput on image processing is used to predict the lifetime of complex agent tasks. Of course the performance of a server should be evaluated only when the server is unloaded. This implies that before the execution of the measurement routine on a server, no applications are running apart from the system processes and the CPU utilisation of the server is zero.

6.3.2.3. Estimating server bandwidths

The network bandwidth between the available servers is determined based on the management agents' message exchange. Every message that is exchanged between the management agents (of every server) is time-stamped. In this instance, the bandwidth between two servers is estimated by dividing the transmission time of a message (from the sender server's agent to the receiver) by the amount of data transmitted. On initialisation every management agent exchange its local system state information (using the SPACE) with the rest of the management agents; the transmit of these messages contributes to the determination of the initial network bandwidths between the servers. The bandwidth between two servers is updated only if there is a significant deviation between the last known recorded bandwidth of those servers and the bandwidth derived from the time-stamped message exchanged between the corresponding sender and receiver agent (on future message exchange). The network latency between two servers is also derived by the management agent based on the utilisation of the "ping" utility. On initialisation every management agent pings the rest of the management agents (their servers) and posts

this information to the SPACE. This procedure is followed on a regular basis.

6.3.2.4. Prediction of the agent's task lifetime

As mentioned previously, the agent tasks are divided into simple and complex[4]. The lifetime of a simple agent task is impossible to estimate beforehand. Therefore the model works on predictions by using the average task completion time of previous accomplished simple agent tasks of the server on which the new task will be initiated. On initialisation the average task completion time is equal to:

$$\mu = \frac{L}{l_a}$$

Formula 6.3. Average task completion time on initialisation

where, L is the processing power of the corresponding server and is divided by a factor l_a which corresponds to the amount of agent tasks launched by the routine that calculated the processing power of that server, in our case 10. Basically, it is the total task time L required by l_a agents to accomplish their tasks. Once an agent has finished its task, its lifetime is used to update the average task completion time which is then calculated based on the following formula:

$$\mu = \frac{(\mu_{old} \cdot a_{total}) + T_{task}}{a_{total} + 1}$$

Formula 6.4. Average task completion time after initialisation

where:

μ_{old} = the previous μ (the lifetime of the first agent sets the initial value of μ_{old})

a_{total} = the number of agents used in the evaluation of μ_{old}

T_{task} = the lifetime of the agent task that has just completed

The prediction of an agent filtering task (i.e. the second part of a complex task) lifetime is mainly based on the amount of data (retrieved from a simple task) that have to be processed. Other factors like the processing power of the compute server on which the filtering will take place, the algorithm that will be

[4] *A simple agent undertakes the acquisition of data composed of a collection of SAR images defined by specific coordinates. A complex agent task may be considered as an extension of a simple one since it requires the filtering of the results acquired by a processing algorithm. A list of all possible cases of an agent's task is in section 6.3.2.5.*

used to process the data etc. that influence the lifetime of the complex agent task are discussed in detail in *section 6.3.2.6 - Case 3 and Case 4.*

6.3.2.5. The model

LB decisions are based on a model which accepts as input an agent's requirements and the system state information, and gives as output the appropriate server(s) to where the particular agent should migrate in order to fulfill its task. Since LB is controlled by the management agents, the model is maintained by UMA and LMA. The model is a function of the following factors: (1) agents' tasks, (2) servers' utilisation (work load), (3) availability of resources at the server, and (4) network efficiency. The model may be better expressed with reference to an agent task, depicted in Figure 6.5. The figure represents in a tree structure all the possible agent tasks. For each of these cases indicated by numbers 1-7, the itinerary of a mobile agent is constructed based on the model factors stated above.

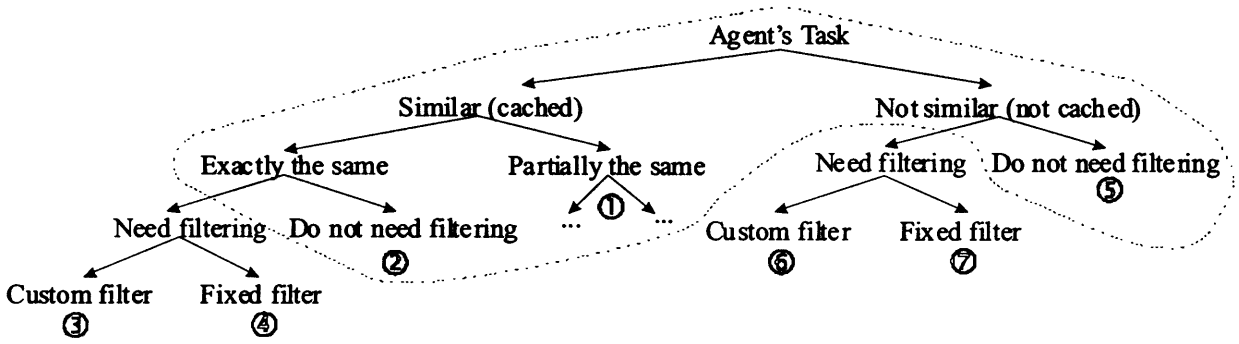


Figure 6.5. Representation of all possible cases of an agent's task in a tree structure

As mentioned in previous sections, an agent task might be simple or complex. A simple agent task which undertakes the acquisition of data composed of a collection of SAR images defined by specific coordinates may be either completely new, exactly the same or similar (part of it) to a task performed by another agent in the past. The coordinates of the images that have to be collected contribute in comparing simple agent tasks. A complex agent task may be considered as an extension of a simple one since it requires the filtering of the results acquired by a processing algorithm that exists on a compute server, referred to as a fixed filter, or by a custom one provided by the user.

The itinerary of an agent is constructed by its local management agent each time before the initiation of

its task. The itinerary of an agent with a simple task comprises a list of database/archive servers with the appropriate resources in descending order based the utilisation of those servers which can serve the agent's task. The first server on the list is characterised as the ideal one, where the agent can accomplish its task faster. The remaining servers provide alternative locations i.e. in case of a failure or overload the agent has an alternative option of migration.

Since the acquisition of information precedes its filtering, the construction of an agent's itinerary with a complex task requires a management agent's support twice. Initially, an itinerary composed of database/archive servers is created for the acquisition of the appropriate information, as in a simple agent task, subsequently a second itinerary for the processing of the data (after they have been collected) consisting of a list of compute servers is necessary. The existence of two separate itineraries is compulsory. First, because it is impossible to decide on which compute server a filtering task can be performed, as the amount and kind of data to be processed is unknown. Second, in a dynamic environment where server/resource conditions change frequently, decisions on load balance must be taken before the initiation of a task.

The most important factor in the model is the utilisation of a server i.e. its load. The construction of an agent's itinerary with a simple task is mainly based on the current utilisation of the available servers, whereas the itinerary of an agent with a filtering task (since its lifetime can be estimated) is mainly based on the predicted utilisation of the available servers i.e. the utilisation of the servers that would result after the execution of the particular task on each of them.

Cases 1 to 5 (Figure 6.5) occur when an agent's task is similar to a task performed by another agent in the past, whereas cases 5 to 7 occur when a task has not been previously performed. In reality, the itinerary of an agent whose task falls into cases 6 or 7 is constructed exactly the same as in cases 3 and 4 accordingly.

6.3.2.6. The different agent task cases

The construction of an optimum itinerary of an agent is based on its task (for each case). This itinerary includes the server(s) to which the agent should migrate to, to accomplish its task. This migration also

supports load balancing. For instance, in the event where there are two available servers (with the required resources) to serve an agent task, the agent will be assigned to the least loaded one. In order to reduce the complexity of the model, results from previous 'cases' may be used; therefore for better understanding it is preferable to read the model from case 1 to case 7. Finally, only simple agent tasks are used for comparison.

Case 1: Agent's task → Similar (cached) → Partially the same.

In this case, the agent's task is partially similar to a task performed by another agent in the past. The 'partially' term used in this context may be better comprehended with the following example. Let us assume that there are two agents with tasks T_a and T_b, where T_a has already been accomplished and T_b is partially the same as T_a i.e. part of T_a's data results exist in T_b's. Since the primary goal of an agent's task is to gather a collection of images of a particular area, agents' tasks can also be reckoned as sets (of

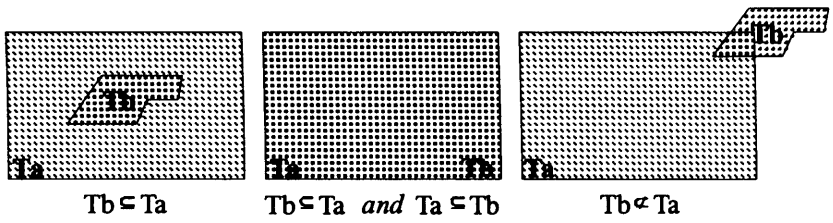


Figure 6.6. Agents' task represented as mathematical sets

images). Therefore, an agent's task is possible to be viewed as a sub-set of another agent's task. That is because mathematically an area (region) can be a sub-set of another area. An area in this instance corresponds to an arbitrary polygon which is defined using a set of coordinates pairs. This case takes into account only the case where T_b is a sub-set of T_a ($T_b \subseteq T_A$), see Figure 6.6. For instance, an example that belongs to this case is the following one: if task T_a concerns the acquisition of information (images) of the area of London and task T_b is regarding the surrounding area of the Big Ben, then $T_b \subseteq T_a$.

The dots under "Partially the same" node of 'case 1' in Figure 6.5, denotes that more nodes exist under that node which are identical to the children nodes of "Exactly the same" node i.e the sub-tree with parent node "Partially the same" is similar to the one that has the "Exactly the same" as a parent node. Therefore, for the agents' tasks that fall into the sub-cases of "Partially the same" case, their itinerary is calculated based on the same logic used in the "Exactly the same" cases i.e. 2,3 and 4. For instance, in

the event that the server that holds an agent's task results (which is a sub-set of a new agent task) is available, as in "Exactly the same"-case 2i, the agent should retrieve the results directly from that server; since the time to extract information from a file is significantly less in contrast with executing an ordinary agent task which involves agent interactions, migrations, utilisation of information resources etc.

The only difference between "Exactly the same" and "Partially the same" cases lies in the corresponding "Exactly the same"-case 2ii of "Partially the same", where the T_{ts} time[5] used in formulas of case 2ii has to be replaced by the amount of time the agent would need to accomplish its task - which is just a sub-set (partially the same) of a previous agent's accomplished one.

This time can be estimated with the use of integrals[144] and algebraic analogy, since the time of a completed agent task - maintained by UMAs - and the coordinates of the areas that enclose the images denoted by both of the agents' task requirements (an already completed task and a sub-set of it) are known. The estimation of this time does not correspond to the exact time required by the agent to accomplish its task, but to a predicted one. This is because a sub-set of an area does not always occupy similar levels of image volume density with a different sub-set of a given area. However, error-estimations may be overcome due to the adaptability algorithm of the proposed model discussed in *section 6.4* of this chapter.

Case 2: Agent's task → Similar (cached) → Exactly the same → Do not need filtering.
In this case, the agent's task is exactly the same as a task performed by another agent in the past and the results do not require any filtering. Since the results of the agent's task have already been stored on a server, referred to as S_1 (at a known location), the advantage of the agent retrieving the results directly from the physical location in contrast with executing the same task on another server, is twofold. First, the agent will provide the results to its user faster; the time needed for the agent to accomplish its task is just to construct a URL pointing to the results where they have been stored. Second, because the agent does not have to use any resources to accomplish its task, it does not actually affect the server's

[5] *T_{ts} denotes the time required by a previous agent to accomplish a task which is exactly the same with a task of a new agent.*

utilisation. In addition, it does not influences the load balance, since the time to accomplish its task is insignificant (almost zero) in conjunction with the time to execute the same task from the beginning.

In the event where the agent's task is the same as the task of another agent which is still in progress, the former agent is deactivated until the later agent finishes its task. UMA management agents maintain information about the progress of every agent task. Therefore, after the later agent finishes its task, the former agent is instructed by its local UMA to activate itself (from persistent stage) and obtain the results directly from the location where the later agent has stored them.

However, investigation is needed in the occasion where server S_1 on which the agent's task results are stored is unavailable. Figure 6.7 presents the different cases that need to be examined, according to the status of server S_1.

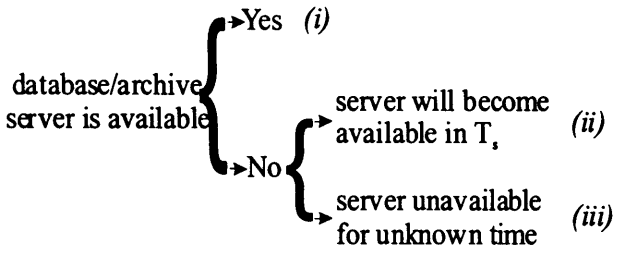


Figure 6.7. Sub-cases of 'Case 2'

The T_s in case (ii) depicted on the above figure may be obtained by system administrators in situations where the period of time required for a server to become available is known e.g. when a server is temporary unavailable due to maintenance, but the time of being back in operation again is predefined.

case 2i: In this case, the server on which the results have been stored is available/pending (online). Hence, the agent should retrieve the results directly from their physical location, as described above.

case 2ii: In this case, the server might be currently unavailable but it will become available in T_s time. Consequently, the point is to check if it is worthwhile for the agent to be deactivated on the server where it currently resides and move to server S_1 when it will become available or if it is preferable to migrate to another server and execute its task; the task that has already been accomplished on the unavailable server

S_1 by another agent in the past. In order to decide if the agent should become persistent or not, T_s should be compared with the minimum time needed for the agent's task to be accomplished on another server, denoted by T_{min}, which is calculated as follows:

Let us denote:

T_{ts} = the time needed by a previous agent to accomplish the same task in the past, on the currently unavailable server S_1.
U_{ps} = the server's S_1 utilisation after the task has been accomplished.
U_s = a server's utilisation.

Based on the information maintained by UMAs on URAs' personal details, it is known that the agent's task to be accomplished by another agent on the currently unavailable server S_1 with known U_{ps} required T_{ts} time. Since a machine of speed 2 should generate twice as many jobs as a machine of speed 1[28], it can be estimated how long it could take an agent to accomplish the same task on a different server of a given U_s. Therefore, for each of the available servers with the appropriate resources the model calculates:

$$x = T_{ts} \cdot \frac{U_s}{U_{ps}}$$

Formula 6.5. Calculation of T_{min}

and for the rest of the servers with the appropriate resources that are unavailable but will become available in T_s, the T_s time of each server has to be added to formula 6.5. Note that the unavailable servers, when they become online, will have a utilisation of zero, since they will have no agents to perform. In addition, for each of the servers evaluated in Formula 6.5, the time the agent needs to migrate itself from its current position to that server also has to be added. For the unavailable servers it is assumed that the bandwidth between the server where the URA agent currently resides, and the unavailable server is equal to the last known bandwidth recorded by the management agent.

The server with the minimum value of x is referred to as server S_2, and T_{min} is equal to that x. If T_{min} is greater than or equal to T_s then the agent should become persistent and wait for the unavailable server S_1 to become online, in order to acquire the results for its task directly from the physical location where they have been stored. The agent by serving its request on S_1 will acquire its results faster, and it will not

influence the load or S_2's utilisation, since it will not consume any of its resources. Even if T_{min} is equal to T_s, although the agent will acquire the results of its task on the same time, regardless to which server it will migrate to (S_1 or S_2), it is preferable to move to S_1 where no resources are expected to be used.
Finally, if T_{min} is less than T_s the agent should become persistent unless this difference is greater than a threshold value q (assuming it is greater than $T_{min}+(T_{min}/2)$) so that T_{min} is much less than T_s. In this instance, although the agent by migrating to server S_2 will influence the server's load and utilisation, it will accomplish its task quicker, in T_s-q time (which is much less than T_s i.e. the time needed for server S_1 to become online). This ensures that agents are also equally served.

case 2iii: In this case, server S_1 is unavailable and it is unknown for how long it will remain offline. Therefore the agent will assume it is a new task, analogous to Case 5.

The management agent that constructed the URA's itinerary keeps a record of the agent's ID accompanied by the IP of the unavailable server that has the results of the agent's task, and the location where they have been stored (by another agent in the past). Once the unavailable server becomes online the management agent sends a message to the URA. If the URA receives the message before it has finished its task, it stops its execution and retrieves the results directly from the (previously server unavailable) location indicated in the message sent by the management agent; otherwise the message is ignored.

Case 3: Agent's task → Similar (cached) → Exactly the same → Need filtering → Custom filter.
In this case, the agent's task is exactly the same as a task performed by another agent in the past, but the results stored by the later agent require filtering by a custom processing algorithm provided by the user. This case pre-supposes that the agent's (task) results have already been gathered, and concerns the creation of the agent's itinerary which is composed only of compute servers. Figure 6.8 presents the different cases that need to be investigated for the selection of servers that can handle the agent's filtering task, based on the status of the compute server collocated at the information-server on where the agent results are maintained.

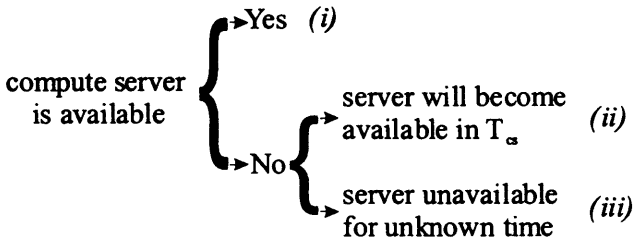


Figure 6.8. Sub-cases of 'Case 3'

The T_{cs} in case (ii) depicted on the above figure may be obtained by system administrators on situations where the period of time required for a server to become available is known e.g. when a server is temporary unavailable due to maintenance, but the time of being back in operation again is predefined.

case 3i: In this case, the compute server where the results have been stored is available. If not always, most of the time the agent completes its task faster when it filters the results acquired by a database/archive server on the compute server at the same information-server. But it should be noted that there is a slight chance that an agent might accomplish its filtering task faster by migrating the data to be processed on another compute server with a lower utilisation than the compute server at the information-server where the results (from the database/archive server retrieved) reside.

In order to find the server where the agent will be served fastest, a prediction of the utilisation of every compute server after the execution of the agent's filtering task would have been performed on each of them, has to be made. The server with the lowest utilisation denotes the ideal one, therefore:

Let us denote:

T_{filter} = the total time required to perform a filtering task
$S_{a.code}$ = the file-size of the URA code.
S_{data} = the file-size of the results (images) retrieved by URA to be filtered.
S_{filter} = the file-size of the custom processing algorithm.
U_s = a server's utilisation.
P_{comp} = the amount of data processed/filtered per millisecond on a compute server.
B_1 = the bandwidth between the web-server where the custom processing algorithm has been stored, and the information-server on which the filtering will take place.
B_2 = the bandwidth between the information-server where URA's results are stored, and the information-server on which URA will have to migrate to filter the results.

An estimation of the predicted utilisation of a server after the execution of an agent's filtering task is expressed by the following formula:

$$U_{predicted}=U_s+\frac{T_{filter}}{L}$$

Formula 6.6. Prediction of utilisation after the execution of a filtering

Basically, the total time of an agent's filtering task T_{filter} is added to a server's utilisation as extra time to its current agent load $(a \cdot \mu)$, that has to be divided by L due to Malone's formula of utilisation: $(a \cdot \mu / L)$. The time required for processing an agent's data against an algorithm is calculated according to the corresponding compute server's throughput. Note that a compute server's performance corresponds to its maximum throughput measured when unloaded, see *section 6.3.2.2*. Consequently, although the processing of data may be expected to last longer on a server of low throughput, its utilisation may end up to be lower (better) than another server of higher throughput if the agent load of the latter is higher than the former.

For the compute server collocated at the information-server, where the database/archive server (from which the agent has acquired its task results) resides, T_{filter} is a function of the predicted time needed to filter the data (S_{data}/P_{comp}), and the time for the custom processing algorithm to be transferred to the compute server (S_{filter}/B_1) where the filtering will take place. For the rest of the available compute servers, the mobile agent's migration time along with the time needed to transfer the results (data) from the server where they have been originally stored to the compute server where the filtering will take place i.e. $(S_{data}+S_{a.code})/B_2$, must also be included in T_{filter}.

For the compute servers which are currently unavailable but will become online in T_{cs} time, the time required for the agent's filtering task to be performed on each of these servers (once they will became online) plus their T_{cs} time, has to be compared with the least time required by the same task to be completed on an available server. If the former outcome is less than or equal to the later one, then it is preferable for the agent to become persistent and wait for the corresponding unavailable compute server to become online in order to perform its task there. Once an unavailable server becomes online it will be

unloaded. Otherwise the server with the predicted lower utilisation indicates where the filtering task will be accomplished faster. For the compute servers which are currently unavailable, it is assumes that bandwidths B_1 and B_2 are equal to the last known bandwidths recorded by the management agents i.e. when the compute servers were online.

case 3ii,iii: In these two cases, the compute server collocated at the information-server where the results have been stored is currently unavailable, and will either remain unavailable for unknown time or become online in T_{cs} time. The selection of compute servers for the accomplishment of the agent's filtering task is done similarly to case 3i. But, for the unavailable compute server collocated at the information-server (where the agent results are maintained) which is known of when it will become online, the data migration time is omitted from T_{filter} function; since the agent's results are on that server. Finally, in case 3iii where the compute server will remain unavailable for an unknown period of time, only the computer servers that are either online or will become available in T_{cs} time will be considered in the list of server that can handle an agent's filtering task.

Note that in case of a failure on the database/archive server where the agent's task results have been stored, before the filtering task is initiated or during its execution, the agent should either wait for the appropriate server to become online or re-compute its data from the beginning according to case 2.

Case 4: Agent's task → Similar (cached) → Exactly the same → Need filtering → Fixed filter.
In this case, the agent's task is exactly the same to a task performed by another agent in the past, and the results stored require filtering by a fixed[6] filter. This case is similar to case 3 apart from the fact that the agent's results need further processing against a fixed filter that exists on a compute server and not against a custom one that has to be downloaded from a user. Consequently, the time required for the custom processing algorithm to be transferred from a web-server to an information-server used in case 3 in the calculations of this case is excluded. Moreover, the list of compute servers that needs to be examined concerns only those servers with the required resources.

[6] *An image processing algorithm that exists on a compute server is referred to as a 'fixed' filter, but 'custom' when it is provided by a user.*

Case 5: Agent's task → Not similar (not cached) → Do not need filtering.
In this case, the agent's task is not similar to a task performed by another agent in the past, where the data that will be gathered does not require any further processing. The decision regarding where the agent should migrate to in order to fulfill its task is based on the utilisation of the servers. The available server with the appropriate resources and the lower utilisation will be the one that will serve the agent faster. Though, there is a possibility that an agent might accomplish its task faster on a currently unavailable server which will become available in a short time, since the server's utilisation on its initialisation will be almost zero. This would be possible if the time for accomplishing the agent task on the currently unavailable server plus the T_s time needed by the corresponding server to become online were less than the time of performing the same task on an available server with the lower utilisation. Because the time of completing successfully an agent task is unknown, the management agents base their estimations on the average time of previously accomplished simple agent tasks.

Case 6, 7: Agent's task → Not similar (not cached) → Need filtering → Custom/Fixed filter.
In these two cases, the agent's task is not similar to a task performed by another agent in the past, where the data that will be gathered requires processing against a custom/fixed filter. Since the acquisition of information precedes its processing, the first itinerary of the agent composed of database/archive servers is constructed based on case 5, where the second one composed of compute servers is constructed based on case 3 or 4 according to the image processing algorithm required i.e. fixed or custom filter.

6.3.2.7. Assumptions of the model

The development of SARA prototype as well as the model of load balance is based on the assumption that an information-server consists of a single machine capable of providing both computational resources and data repository facilities. This can be better apprehended as an information-server having two virtual servers. When the computational resources of an information-server are unavailable, it is assumed that its compute server is down, although its ability to provide data repository facilities might be available.

As a result the migration time of an agent or object from the database/archive server to the compute server of the same information-server is null, since it is referred to the same machine. This implies that

an agent's results stored on a database/archive server can directly be filtered (locally) against an image processing algorithm if the corresponding information-server provides the appropriate computational resources.

Another assumption concerns complex agent tasks. The filtering of an agent's task results against a fixed filter may be performed only by the compute servers possessing the required filter, whereas a custom filter can be transferred and be executed on any available compute server.
In addition, as Malone's formula of utilisation requires the numerator to be non-zero, even if a server is unloaded it is assumed that there exists one agent with an average task time of 0.1 milliseconds. Of course this agent is excluded from the calculations on the average task completion time of agents on a server. Furthermore, the persistent agents are not taken into account since they do not consume any vital resources. In the Voyager platform the persistent agents are considered as object of a database[156] and the only processing power they require is during their transaction from active to persistent and vice-versa.

Moreover, the majority of agent tasks initiated in SARA involve simple tasks with small variation on their lifetime. Changes in the lifetime of complex tasks vary analogous to the amount of data to be filtered, but they have a great affect on the utilisation of a server in comparison with simple tasks. Finally, the information regarding the URA agents maintained by the UMA management agents are kept until the user deletes these files i.e. the results gathered from his/her representative URA agent or during the LAA agent's file-space maintenance check, see *Chapter 7 - section 7.2.1.1.1.*

6.4. Adaptability of model

The Enterprise and Challenger model-based approaches to LB discussed in *Chapter 3*, use Malone's formula of system utilization, and their model is based on the distribution of CPU load and expected lifetime of tasks.

However, there are tasks for which lifetime is impossible to be estimated beforehand (e.g. the time a user is running a remote application) or their time duration estimated by users or special routines are erroneous. To deal with such error estimations, Enterprise system uses an *estimation error tolerance* parameter. If a task takes significantly longer than it was estimated to take (i.e. more than the estimation

error tolerance), the server running the task aborts it and notifies the user which initiated the task that it was *cutoff*. This cutoff feature prevents the possibility of a few people or tasks monopolising an entire system. Challenger[28] on the other hand introduces learning behaviour in the bidding agents to deal with errors in estimating task completion times. The idea is based on *penalising* those agents which misestimate the lifetime of their tasks. Therefore, during a bid evaluation process, each agent's bid (i.e. lifetime of its task) is adjusted by multiplying it by the agent's current inflation factor. For instance, if an agent has recently been making perfectly accurate bids, its inflation factor will be 1.0 and its bid will not be altered. Otherwise, if an agent has been recently turning in task completion times that are twice as slow as what it estimated, then its bid will be multiplied by an inflation factor of approximately 2.0.

The SARA model is based on simple agent tasks for which the lifetime is predicted to be equal to the average task completion time of previous agents on a given server, and on filtering tasks (i.e. the second part of a complex task). Lifetime can be estimated based on calculations on the collected data to be filtered. The major parameter used in distributing tasks among the servers is the utilisation of the available servers. If the lifetime of filtering tasks was unknown then the model would not function properly, since filtering tasks influence the utilisation of a server significantly more than simple tasks, because they require more processing power and time. Therefore, in order for the SARA model to be applicable to other systems where lifetime of complex tasks is impossible to estimate or predictions on lifetime of tasks are erroneous, the model should provide a means of self-adapting to such error estimations. The policy of the Challenger system on penalising the agents for misestimating the lifetime of their tasks based on prior recorded estimations cannot be followed by SARA model, because in SARA each user request (task) is represented by a different agent. The approach of the Enterprise system involves setting a threshold value, which when exceeded, causes the task to be terminated is impractical for tasks for which their lifetime is unknown.

The adaptability of the SARA model is based on an algorithm for systems where the lifetime of complex task cannot be estimated. The algorithm is activated by the management agents and its objective is to monitor the utilisation of every server, and amend the model when found to be miscalculated, due to the introduction of agent tasks with unknown lifetime in the servers.

6.4.1. Description of Algorithm

Load balancing in SARA is based on the utilisation of servers i.e. every task is assigned to the server with the available resources (according to the agent's task demands) and the lowest utilisation. The utilisation of a given server has a direct relation to the average task completion time of the agents on that server. Though, the utilisation of a server is only updated when the server's agent load changes i.e. when an agent enters or leaves the server. Since the lifetime of complex tasks is unknown, the selection of servers on which agents can fulfill their tasks is based on the current utilisation of servers with the available resources, as with the simple tasks, and not on the predicted utilisation that the servers would have after the execution of a complex task (this is infeasible due to the unknown lifetime of complex tasks). This implies that the utilisation of a server on the arrival of a complex task is not actually affected, since the lifetime of the corresponding complex task is not added as extra time to the server's agent load, resulting in incorrect evaluation of a server's utilisation.

The algorithm depicted in Figure 6.9, runs on each server separately. On the arrival of the first agent on a server, the algorithm sets a timer. After a predefined time a procedure called *check_AvTaskComplTime* is executed. Initially the timer is set equal to the average task completion time of agents on the server, derived by the routine of a server's processing power estimation (see section *6.3.2.4. Prediction of the agent's task lifetime*).

The *check_AvTaskComplTime* procedure basically monitors the transit of agents on a server. If no agent has left the server up to the time when *check_AvTaskComplTime* has been initialised, it means that the number of agents on that server has either increased or remained unchanged. This implies that the agents on the server (or even the first agent that arrived on the server) have not accomplished their task on time i.e. within the time corresponding to the average task completion time of an agent. Since the average task completion time of agents on a given server (its utilization) are updated only after the departure of an agent from that server, utilisation is not updated until an agent completes and departs.

Provided that the agents require more time to complete their task, the algorithm's objective is to update the utilisation of that server based on the increase in the average task completion time of those agents and publish this information to SPACE. This utilisation is an estimate, and is posted to the SPACE in

regular intervals until an agent departs from the server. Once an agent leaves the server, the lifetime of its task is used to update the average task completion time of agents. The timer of the algorithm which activates the *check_AvTaskComplTime* procedure is triggered in different time intervals, due to the change in the average task completion time of agents throughout time, and it is disabled when there are no agents on server.

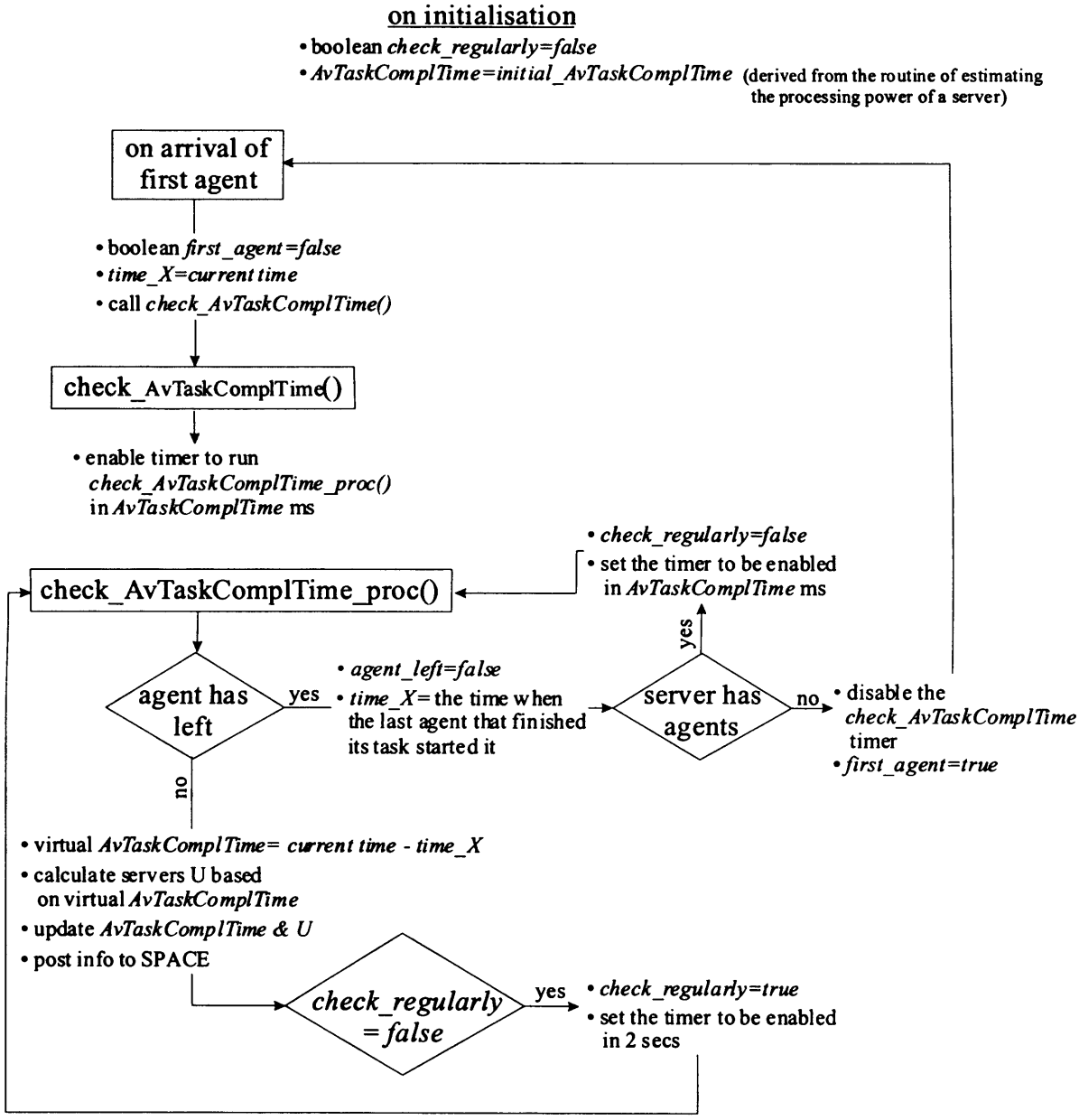


Figure 6.9. Adaptability algorithm of SARA LB model

Experimental results, see *Chapter 8*, show that on a system of which the lifetime of its complex tasks is unknown, the utilisation of the special algorithm provides an optimisation of 1.63% to 10.8% in load balancing, analogous to the amount of complex tasks introduced (5%-25% accordingly).

To conclude, the design of a LB model depends on the properties and functional needs of the agent-based system. The SARA model may be employed by other agent-based systems utilising active archives, in situations where the lifetime of complex tasks cannot be estimated or tend to be erroneous. In such systems, developers can take advantage of the adaptability of the model provided by the algorithm described.

For instance the ALABAMA[82] algorithm of the FLASH system, by which the state-based part of SARA LB has been inspired, is impractical for such systems since it focuses on providing solutions to large scale problems. The algorithm includes two phases. The first phase involves the rearrangement of mobile agents in the system for achieving a basic even distribution of mobile agents among the available servers, which results in a large number of migrations. In the second phase the load is improved by performing fewer migrations based on system state information used by the mobile agents. The algorithm assumes homogeneous resource demands of applications, and does not consider network connection characteristics, where observations of the system performance shown that the efficiency of the ALABAMA strategy depends on the quality of the first phase.

6.5. Conclusion

This chapter has presented a load balancing mechanism to enable specialised stationary agents to gather system state information and make decisions on the distribution of mobile agents among the servers. This is based on a model of predictive estimations in relation with the information provided by the stationary agents. There will undoubtedly be errors in the estimation of the model but due to the information on the progress of the URA agents and the observation of the whole system provided by the UMA management agents, it is possible to optimise the intelligence of the management agents for improving their accuracy on load balancing decisions. For instance, based on statistical information generated by the management agents the effect of an agent migrating to a server, the lifetime of the agents executing on that server can be deduced and contribute to the amendment of the model.

Chapter 7. Implementation

7.1. Introduction

A detailed description of the SARA prototype (constituted of 63 Java Classes, 11,365 lines of code in total) cannot be provided within a few pages, though this chapter presents the structure of every entity involved in the system and discusses the most important implementation issues. The chapter is divided in four main sections. The first section presents the software applications required for the development of the prototype, the client-side and server-side are covered in the second and third sections accordingly, and implementation considerations are discussed in the fourth section.

7.2. Implementation of SARA prototype

The main software required for the implementation of the prototype is the Java programming language, Voyager agent platform and FIPA-OS toolkit. Java[89] was chosen as the programming language on the development for the prototype because of its many attractive features, particularly geared towards object-oriented programming in distributed heterogeneous environments, platform independence, object serialisation, multithreading, Remote Method Invocation (RMI), secure execution and dynamic Class loading which are essential for implementing a mobile agent system.

Voyager[154] of Recursion software Inc. was chosen as the environment within which agents can be created, interact, migrate and communicate between themselves due to its feature-rich, reliable, easy-to-use platform and its ability to design and deploy robust, distributed enterprise applications[155]. Comparison reports [6][21][34][43][126][137] show that the Voyager agent platform performed better than most other commonly used platforms; like Grasshopper[72], Aglets[3], Jade[85], Zeus[170], Concordia[158] and Mole[105]. A comparison between RMI and Voyager showed that for remote object creation Voyager was 25% quicker than RMI, whereas RMI outperformed Voyager by an average factor of 2 for remote object connection, remote method calls and object array transfer[80]. For transfer of byte, integer and double arrays Voyager outperformed RMI.

The FIPA-OS[60] component-based toolkit developed by Nortel Networks was chosen for the development of the FIPA-compliant gateways. FIPA-OS enables the rapid development of FIPA

compliant agents and is continuously improved as a managed Open Source community project, characterised as the *ideal* choice for any FIPA compliant agent development activity by the FIPA organisation[57]. In addition, JSDK (Java Servlet Development Kit)[90] and OReily's servlet JDK[112] ware used for the construction and management of servlets; the HTTP-server provided by JSDK was used to handle the static and dynamic HTML-pages of the SARA web-site. The JAI (Java Advanced Imaging)[86] development kit was used for the implementation of fixed filters and the Oracle DBMS[111] was used to maintain the data repository of every information-server.

The prototype may be divided into the server-side and client-side software as depicted in Figure 7.1. The server-side corresponds to a number of information-servers with appropriate data repositories and compute facilities, whereas the client-side provides the front-end to users/agents that need to access the digital library composed of one or more web-servers.

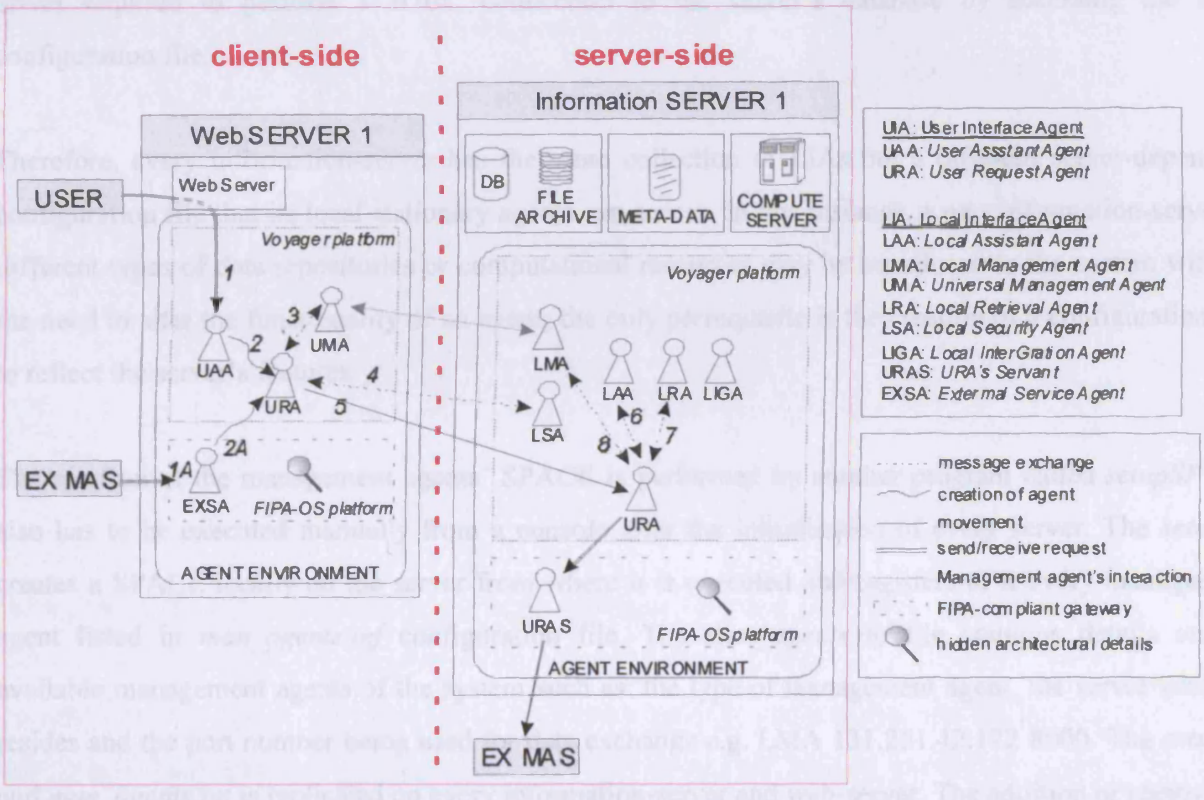


Figure 7.1. SARA client and server side

7.2.1. The server-side

The initialisation of an information-server is performed by the manual execution of a program called *Server* from a console. The program launches the Voyager server, creates a proxy for the local management agent and activates the resource-check of LAA. Every user may have a fixed amount of physical storage on every information-server and one of LAA's responsibilities is to maintain the file-space of each user, discussed in *section 7.2.1.1.1.*

The configuration of an information-server is based on a file (*config.inf)* which can be easily amended with a simple text editor and contains information on the corresponding server. An example of a configuration file in XML form is depicted in Code 7.1. The Local Interface Agents (LIAs) of each information-server have access to that file and can retrieve information related to their tasks. For instance, the LAA of an information-server is aware of the fixed filters provided by the latter or the driver required to perform a JDBC connection to the server's database by accessing the local configuration file.

Therefore, every information-server has the same collection of LIAs but a different server-dependent configuration file that its local stationary agents can access. In this instance, a new information-server of different types of data repositories or computational resources may be introduced to the system without the need to alter the functionality of an agent; the only prerequisite is the creation of a configuration file to reflect the server's features.

The creation of the management agents' SPACE is performed by another program called *setupSP* that also has to be executed manually from a console after the initialisation of every server. The *setupSP* creates a SPACE locally on the server from where it is executed and registers to it every management agent listed in *man_agents.inf* configuration file. The *man_agents.inf* file contains details on the available management agents of the system such as: the type of management agent, the server where it resides and the port number being used for data exchange e.g. LMA 131.251.42.172 8000. The *setupSP* and *man_agents.inf* is replicated on every information-server and web-server. The addition or removal of a server requires the appropriate amendment in the *man_agents.inf* file of each server.

The console window of the *Server* program of each information-server displays its stationary agents' interactions with any incoming mobile agents, its management agent's actions and information exchange along with any error or warning messages that emerge during the agents' execution. Screen-shots of information-server consoles can be found in Figure 8.3 and 8.5 of *Chapter 8* in *sections 8.2.1* and *8.9.2*.

```
<?xml version="1.0" ?>
<server_specifications>
  <Server name="Illinois.cs.cf.ac.uk">
    <IP>131.251.47.212</IP>
    <CPU_type>UltraSPARC-IIi</CPU_type>
    <Processing_power>112597</Processing_power>
    <Filtering_throughput>111244</Filtering_throughput>
    <OS>solaris</OS>
    <Storage medium="hdd"
             type="raid" feature="0,1"
             capacity="420" measurement="Gb"/>
    <Data_archive="relational database"
          type="Oracle"
          release="8.1.6.0.0"
          image_resource="/home/scmcg1/image_resource"/>
    <Type_of_connection="JDBC"
           JDBC_driver="Jdbc:oracle:thin:@delphi:1521:cs2000"
           JDBC_class="oracle.jdbc.driver.OracleDriver"/>
    <Server files="/home/scmcg1/config"/>
    <Analysis_alg>
      <Filter_description="Edge detection (Mexican hat/Marr) - 13x13 matrix"
           name="filter1"
           location="/home/scmcg1/filters">
      </Filter>
      ...
      <Filter_description="Sharp Filter - 3x3 matrix"
           name="filter4"
           location="/home/scmcg1/filters">
      </Filter>
    </Analysis_alg>
    <User_path="/home/scmcg1/public_html"
         URL="http://www.cs.cf.ac.uk/User/C.Georgousopoulos1"/>
         resources="/home/scmcg1/config/users_l.tmp"
         max_file_space="7.5" measurement="Mb"
         delay_check_file_space="36000" measurement="sec"/>
  </Server>
</server_specifications>
```

Code 7.1. Example of an information-server's configuration file (*Config.inf*)

The following sub-sections describe the implementation of every LIA on the server-side developed to offer an extensible set of services to provide a level of abstraction between information-servers and the requesting mobile agents. The functionality of every agent is decomposed into different Java Classes according to the number of services that the agent provides. This partition enables the easy modification of an existing service provided by an agent or the introduction of a new one without affecting the rest of the agent's code.

7.2.1.1. LAA - Local Assistant Agent

The basic role of LAA is to support interaction with any visiting URAs and assist the completion of the task carried by a URA. This involves the connection to the server's data archive performed by LAA on behalf of URA for the execution of its query, and the filtering of data against an analysis algorithm if this is required. LAA also monitors the file-space of each user on the local server. LAA is composed of *Laa_rc*, *Laa_con*, *Laa_discon*, *Laa_proc_alg* and *Laa_cd_alg* Classes. These Classes are now described in detail.

7.2.1.1.1. Laa_rc

Every user has a fixed amount of physical storage on an information-server where the results of his/her requests are being stored. Laa_rc's (LAA Resource Check) objective is to maintain the file-space of every user and prevent someone from exceeding the limit of storage space that s/he owns on the local information-server.

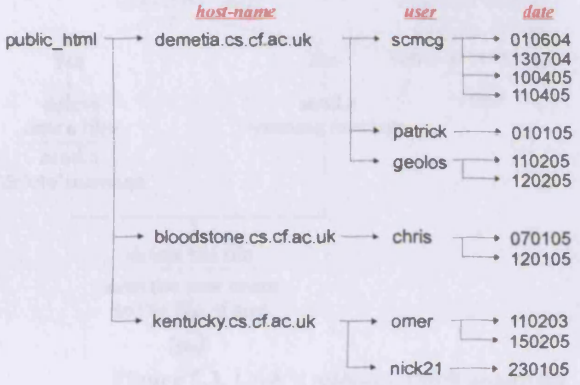


Figure 7.2 Hierarchical directory structure

The LRA executes a user's query on the server's data archive, formats and stores the results in a well-defined hierarchical structure as depicted in Figure 7.2. The *root* directory-name of every user is defined in the server's configuration file (*config.inf* [1]) and has the property of publishing document files to the internet; in this instance it is called *public_html*. The *root*'s child-directory is named by the *host-name* of the web-server on where the user's URA agent is initially created. Its child-directory has the name of the user, and inside the *user* directory there is a leaf directory named by the *date* of URA's creation. This

[1] *An example of a server's configuration file can be found in* Code 7.1 *of* section 7.2.1.

name is in the form of day-month-year (e.g. 110405), derived by the clock of the corresponding web-server. The user's files are stored in that directory i.e. *date* directory.

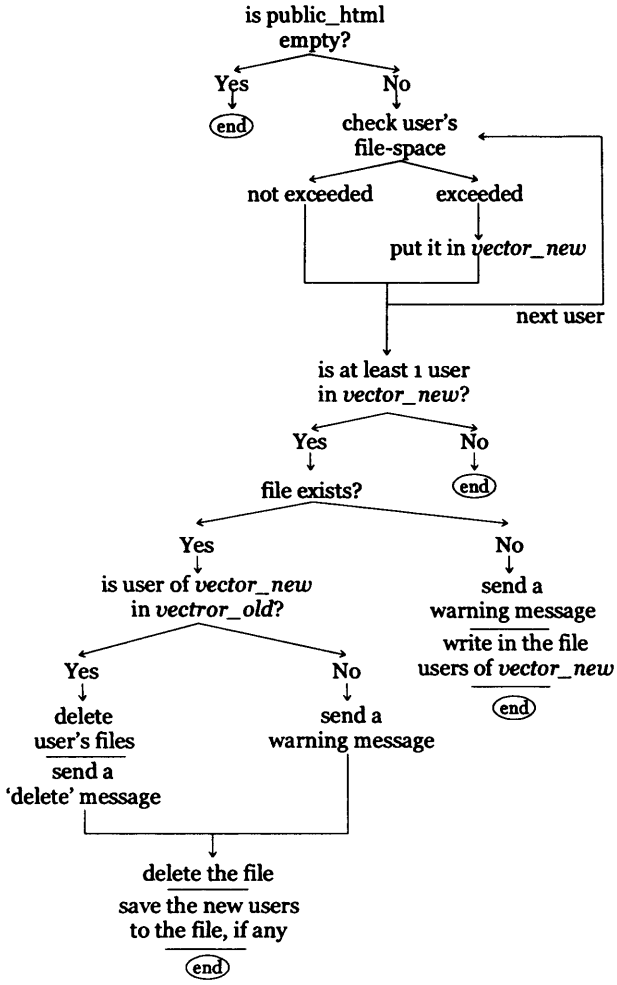


Figure 7.3. LAA's resource-check algorithm

LAA performs a resource-check on a regular basis, as specified in the configuration file by the "delay_check_file_space" parameter. In the event that a user has exceeded the limit of his/her file-space, LAA informs the corresponding user via UAA or e-mail and a warning-message is displayed on the server console for that user. On the next cycle of LAA's resource-check, if the user has not taken any actions to preserve the limit of his/her file-space, LAA deletes the user's oldest files and displays an appropriate message to the server console. Laa_rc's resource-check algorithm is illustrated in Figure 7.3.

7.2.1.1.2. Laa_con and Laa_discon

LAA along with LRA provides a level of abstraction to an information-server's data repository. LAA's responsibility is to supply URA with information on how to access the data repository of the local server, whereas LRA's objective is to execute a query to the data source on behalf of URA.

LAA is aware of the type of data source maintained by an information-server and the way to access it based on the local server's configuration file, identified by the "Data_archive" and "Type_of_connection" parameters. In the case where the data achieve is comprised of flat files, Laa_con (LAA Connect) provides LRA with the physical location on where to search for, to execute URA's query. However, if the data achieve is maintained by a Data-Base Management System (DBMS), Laa_con has the required code to create a connection to the DBMS necessary for the execution of an SQL statement. After Laa_con performs a successful connection to the DBMS it supplies LRA with the serial-number (ID) of the connection established. Subsequently, the LRA can execute the URA's query after it has been transformed to an SQL statement. The connection to the DBMS is terminated by Laa_discon (LAA Disconnect) on behalf of URA based on the connection ID supplied, after LRA has accomplished its task. Laa_con and Laa_discon are implemented in two different Classes because they represent two separate Java threads.

In Java a database can only be accessed using JDBC (Java Data-Base Connectivity). JDBC is a call-level API that is used to connect and pass SQL statements to a relational database engine. Figure 7.4 illustrates the four possible approaches of accessing a relational database.

The first approach *(type 1)* makes use of the JDBC-ODBC (Open Data-Base Connectivity) bridge driver, where JDBC API calls are translated into ODBC calls i.e. an API that defines the routines for accessing MS-Windows databases e.g. MS-Access, and sends them to an ODBC driver already installed on the server. The second approach *(type 2)* uses a JDBC driver written entirely in Java, where the statements passed to the SQL server have to be first translated by a middle-tier gateway at the server into a DBMS-specific protocol. The third approach *(type 3)* makes use of a JDBC driver written in Java and in native code i.e. Database-Vendor specific code, and the last approach *(type 4)* makes use of a JDBC driver written entirely in Java that access directly the SQL server.

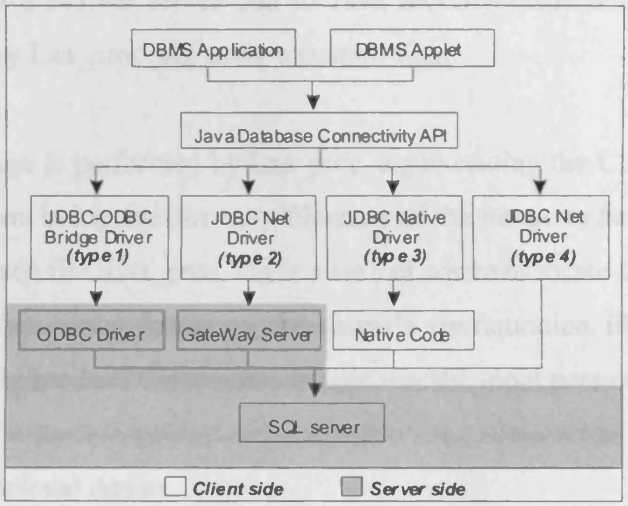


Figure 7.4. JDBC methodologies[20]

Laa_con can connect to any kind of relational database, if the appropriate JDBC/ODBC driver is installed on server, according to the information provided in the information-server's configuration file denoted by the "Type_of_connection" parameter. In the prototype every Laa_con is using a JDBC driver of *type 4* to connect to the database of its information-server maintained by the Oracle DBMS.

7.2.1.1.3. Laa_proc_alg and Laa_cd_alg

The Laa_proc_alg (LAA Process Algorithm) Class of LAA enables the processing of data against a fixed or custom filter. Laa_proc_alg parses the results stored in an XML file (see example in Code 4.6 of *Chapter 4 - section 4.4*) specified by URA and processes every image identified by the element "SARATRACK IDTRACK" in a separate Java thread against a filter. The resulting images are saved in the same directory from where the XML file is accessed.

Laa_cd_alg (LAA CopyDelete Algorithm) is a complementary Class that undertakes the procedure of transferring a custom filter uploaded by a user from a web-server to the information-server where the filtering will take place. Laa_cd_alg creates a URL address pointing to the web-server's publicly accessible directory where uploaded custom filters are stored, reads byte by byte the appropriate file (custom filter) and saves it locally on server. After the file is reconstructed successfully Laa_cd_alg instructs the UAA of the corresponding web-server to delete the file on behalf of it, since LAA cannot

delete a file by itself on a remote server due to Java security restrictions. Subsequently, the image processing is performed by Laa_proc_alg using a custom filter.

The processing of an image is performed by Laa_proc_alg executing the Class file of the corresponding filter with input parameters being the directory/filename of the image to be filtered, and the location of where to store the processed file. Laa_proc_alg is aware of where to locate the Class of every fixed filter provided by its information-server based on the server's configuration file, where the custom filters supported by Laa_proc_alg are Java Classes that accept specific input parameters.

7.2.1.2. LRA - Local Retrieval Agent

As it has been mentioned earlier, part of LAA in collaboration with LRA provide a level of abstraction to an information-server's data repository. LRA is composed of two Classes Lra_EXquery (LRA Execute Query) and Lra_updXML (LRA Update XML). Lra_EXquery receives from URA an SQL query and a serial number of the JDBC connection established to the information-server's database (by LAA on a previous stage) as parameters. The agent executes the SQL statement to the database based on the connection supplied and retrieves the results row by row. As every row of information is received, it is parsed on-the-fly and appropriate XML-tags are introduced according to the Document Type Definition (DTD) of Code 4.5, see *Chapter 4 - section 4.4.*

After the results have been formatted to XML they are saved in a file named by the URA's ID which is unique. Lra_EXquery assigns space to every user, and the file that contains the results of his/her request is stored inside a directory that complies with the hierarchical structure discussed in *section 7.2.1.1.1.* Therefore, even if a user performs more than one request on the same day, every file that corresponds to a single request is stored in the same directory (see Figure 7.2) but with a different unique filename.

If a user request concerns the fusion of results against an image processing algorithm, Lra_updXML generates a new XML file based on the one created by Lra_EXquery to include the metadata for the processed images. The procedure involves the parsing of the original XML file that contains the results of a request and the introduction of appropriate information to describe the resulting images. This

includes the location and names of the processed images, the filter used etc. The name of the XML file is derived from the filename of the original XML with the current time attached at the end.

7.2.1.3. LMA - Local Management Agent

LMA's responsibility is to create the itinerary of a URA based on its requirements and the current system status, balancing at the same time the load of mobile agents among the information-servers. The model of load balancing, discussed in *Chapter 6 - section 6.3.2.5*, is implemented in one of LMA's *methods* that can be accessed by a URA via direct messaging to accept the description of a task, and return an appropriate itinerary for the mobile agent.

On the initialisation of an information-server, see Figure 7.5, LMA creates an empty temporary file locally that is deleted only after a successful termination of the management agent. Therefore, if an information-server fails and is restarted again, the LMA can evaluate that its server was previously down by the existence of the temporary file. LMA contacts the first management agent listed in its *man_agents.inf* file via direct messaging to acquire the SPACE details[2] and the system state information about the rest of the servers, and registers itself to the SPACE. If a management agent is inaccessible it proceeds to the next one in the list. Subsequently, LMA publishes its local system status to the SPACE and the information-server is ready to operate i.e. accept any incoming mobile agents.

LMA has a *method* to publish information to SPACE via multicast messaging, another one to receive and parse the information sent by other management agents, and a different one to receive data from URAs such as the lifetime of their tasks or the IP of a failed server. Any information that is published in SPACE has the name of the SPACE and the IP address of server hosting it attached i.e. the SPACE details.

The name of a SPACE is derived from the time when it is created (expressed in milliseconds); along with the IP address of the server hosting the SPACE a unique ID is formed. Therefore, it is important that the clock of every server in the network is synchronised with a Time Server. On the SARA prototype the clock of every server was synchronised with a Time Server based on the Network Time

[2] *The name of a SPACE and the IP address of the server hosting it.*

Protocol (NTP)[102] of the UNIX Operating System. In addition, two SPACES are not allowed to coexist on a server. A management agent can only create a SPACE on its server and none on a remote one.

The LMA, upon reception of data sent via the SPACE, parses them and updates its information concerning the rest of the management agents. If the SPACE details of a message differs from the one held by LMA it means that the server where the SPACE has been initially created has failed, but a new one was reconstructed by another management agent; the one that was the first to send a message to SPACE and failed. In this instance LMA compares the two SPACE names, stores in its information the oldest one which is *alive* and deregisters itself from the other one. This procedure ensures that a single SPACE i.e. the oldest one will be kept for information exchange, in the event where more than one management agents have created a new SPACE.

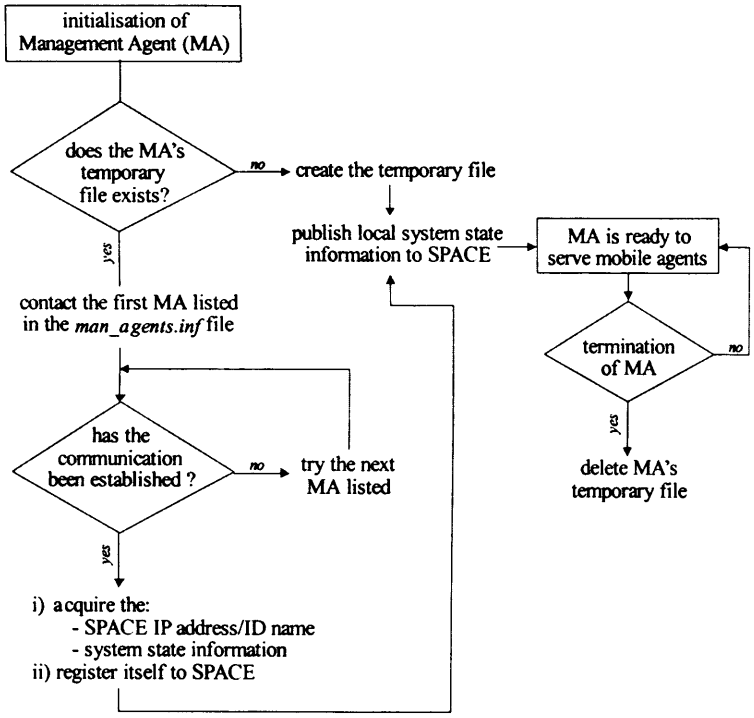


Figure 7.5. Initialisation of a Management Agent

Other *methods* exist for the determination of the network connection bandwidth between the local server and the remaining ones, the update of the average task completion time of local agents, and the server's utilisation.

7.2.2. The client-side

The client-side requires the existence of at least one web-server. The initialisation of a web-server is performed by the manual execution of *WServer*, *startFIPAOS* and *startserver* programs from separate consoles. *WServer* launches the Voyager server, creates a proxy for the local management agent and initialises the EXSA gateway agent. The *startFIPAOS* of FIPA-OS toolkit activates the FIPA-compliant gateway and initialises the AMS and DF to which EXSA must register itself. The HTTP-server of JSDK is enabled from *startserver* to provide a gateway to the web.

The console window of *startFIPAOS* records the interaction of AMS, DF and EXSA with foreign agents, where the console of *WServer* similar to *Server* program displays information on the progress of agents; see Figure 8.2 and 8.5 of *Chapter 8 - sections 8.2.1* and *8.2.2* accordingly. In addition, every web-server maintains a configuration file that local UIAs and LIAs can access to retrieve parameter information. Code 7.2 illustrates an example of a web-server configuration file in XML form.

```
<?xml version="1.0" ?>
<server_specifications>
  <Server host="sapphire:8000"
          www_URL="http://131.251.42.9:8080>"
          www_path="/home/scmcg/project/web_server/"/>
</server_specifications>
```

Code 7.2. Example of a web-server's configuration file

External FIPA-compliant agents access SARA through the EXSA gateway agent, whereas users define their queries via a web-based GUI. The web-site consists of a single static HTML-page with two frames; see Figure 7.6. The top frame contains an HTML-form, and a UAA (implemented as a servlet) collects the information entered by a user. The bottom frame is used to display the content of results created by UAA after the user's request has been accomplished. Details on the implementation of four fixed filters developed (which users can choose to further process the results of their request) using JAI library can be found in *Appendix A4*.

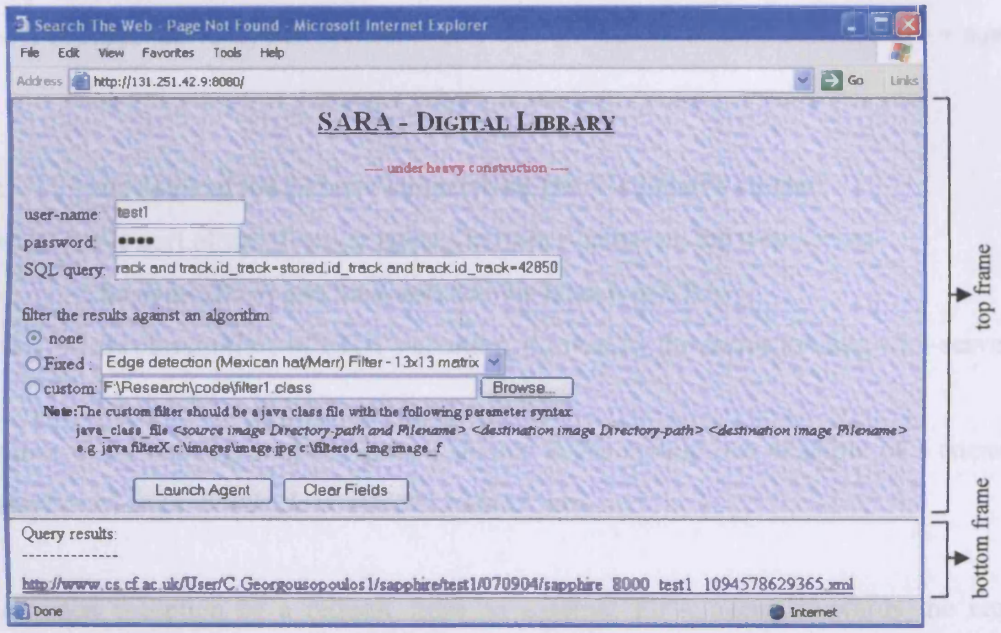


Figure 7.6. The SARA web-page GUI

7.2.2.1. UAA and EXSA

UAA (User Assistant Agent) and EXSA (External Service Agent) may be characterised as intermediary or representational agents since they both accept a user/agent request and deliver the results. UAA is in the form of a servlet consisting of a single Class with different *methods* to:

i) capture a user's information from the HTML-page of the web-based interface,
ii) upload a custom filter from a user's computer to the web-server,
iii) create a proxy of URA, supply it with parameter information obtained from the server's configuration file and the HTML-page (regarding a user's request), and instruct it to initiate its task,
iv) present a user's results upon reception from URA by constructing a dynamic html-page,
v) delete a custom filter on the server-side when it is instructed by LAA.

Every URA that it is dispatched from UAA or EXSA is named by a unique ID derived from the following parameters:

host-name + (Voyager server's) port number + user-name + current time (in milliseconds) + agent's name

where:

host-name = the name of the web-server on where URA is initially created.

port number = the port of the Voyager agent platform running on the web-server.

user-name = the name of the user that initiated the launch of URA

current time = the current time of URA's creation, derived by the corresponding web-server's clock.

In this instance every roaming mobile agent is unique in the system. An example of a correct ID name might be *pearl.cs.cf.ac.uk_8000_chris_943212505437_ura.*

The EXSA, upon reception of a request from an external FIPA agent, forwards the request to its complementary Class EXSA_serv (EXSA Servant) and replies to the external agent with the results of its request after it has been fulfilled by the appropriate URA. EXSA_serv dispatches a URA on behalf of an external agent in the same manner as UAA does. The implementation of EXSA based on the GatewayAgent API and the configuration of the FIPA-compliant gateway has been described in detail in Chapter 5. UAA, URA, LAA and EXSA interact between themselves via *OneWay* direct messaging.

7.2.2.2. URA - User Request Agent

URA is a composed of a single Class with the necessary code to migrate through the information-servers and interact with the stationary LIAs in order to fulfill a request placed from an external agent or a user. URA before the initiation of a task comes in contact with the local management agent of the server where it resides and receives from the latter its itinerary, based on the mobile agent's requirements and the current system state. An itinerary may be in the form of:

i) a list of database/archive servers with the appropriate resources in descending order, based on the servers' utilisation that can serve the agent's task,

ii) a list of compute servers, if the agent's task concerns the filtering of existing data acquired by URA itself on a prior task,

iii) a single URL pointing to the results of the agent's task that has been already accomplished by another agent in the past (having the same task),

iv) a URL along with a list of compute servers, if the results pointed to by the URL require further processing i.e. filtering.

The migration of URA involves the transmission of its code and state from one server to another. If the agent's task requires the filtering of data against a custom filter, LAA on behalf of URA uploads the Class file representing the custom filter to the server where the filtering will take place. Due to the Voyager callbacks that it receives before and after its migration URA is capable of autonomously selecting an alternative server to migrate to in the event of a failure. If such an event occurs, URA informs the management agent of the source server that the destination server is inaccessible and the latter posts this information to SPACE. Voyager provides the callbacks capability through the *IMobile* interface[156]. Generally, an object that implements *IMobile* interface, receives callbacks during a move in the following order:

- preDeparture(String source, String destination)

  This method executes on the original object at the source. If the method throws a *MobilityException*, the move aborts and no more *IMobile* callbacks occur.

- preArrival()

  This method is executed on the copy of the object at the destination. If the method throws a *MobilityException*, the move aborts and no more *IMobile* callbacks occur.

- postArrival()

  At this point the copy of the object becomes the real object, the object at the source becomes the stale one, and the move is deemed successful. This method executes on the copy of object at the destination.

- postDeparture()

  This method executes on the original stale object at the source. It is typically defined to perform activities such as removing the stale object from persistence. Messages sent to the stale object via a proxy are redirected to the new object.

Voyager automatically tracks the current location of an object. If a message is sent from a proxy to an object's old location, the proxy is automatically updated with the new location and the message is re-sent. This is achieved by the use of a chain of forwarding pointers left behind on every server visited by the migrant object, managed by Voyager hidden to the developers.

The interaction of URA with a LIA is performed in a separate Java thread. In this instance, URA can concurrently interact with more than one LIAs and every LIA can provide its services to different URAs simultaneously. The basic algorithm of URA is depicted in Figure 7.7. Migration strategies for mobile agents that are consist of more than one Class can be found in [49], and a performance evaluation of those strategies with regard to network load and transmission time is in [17].

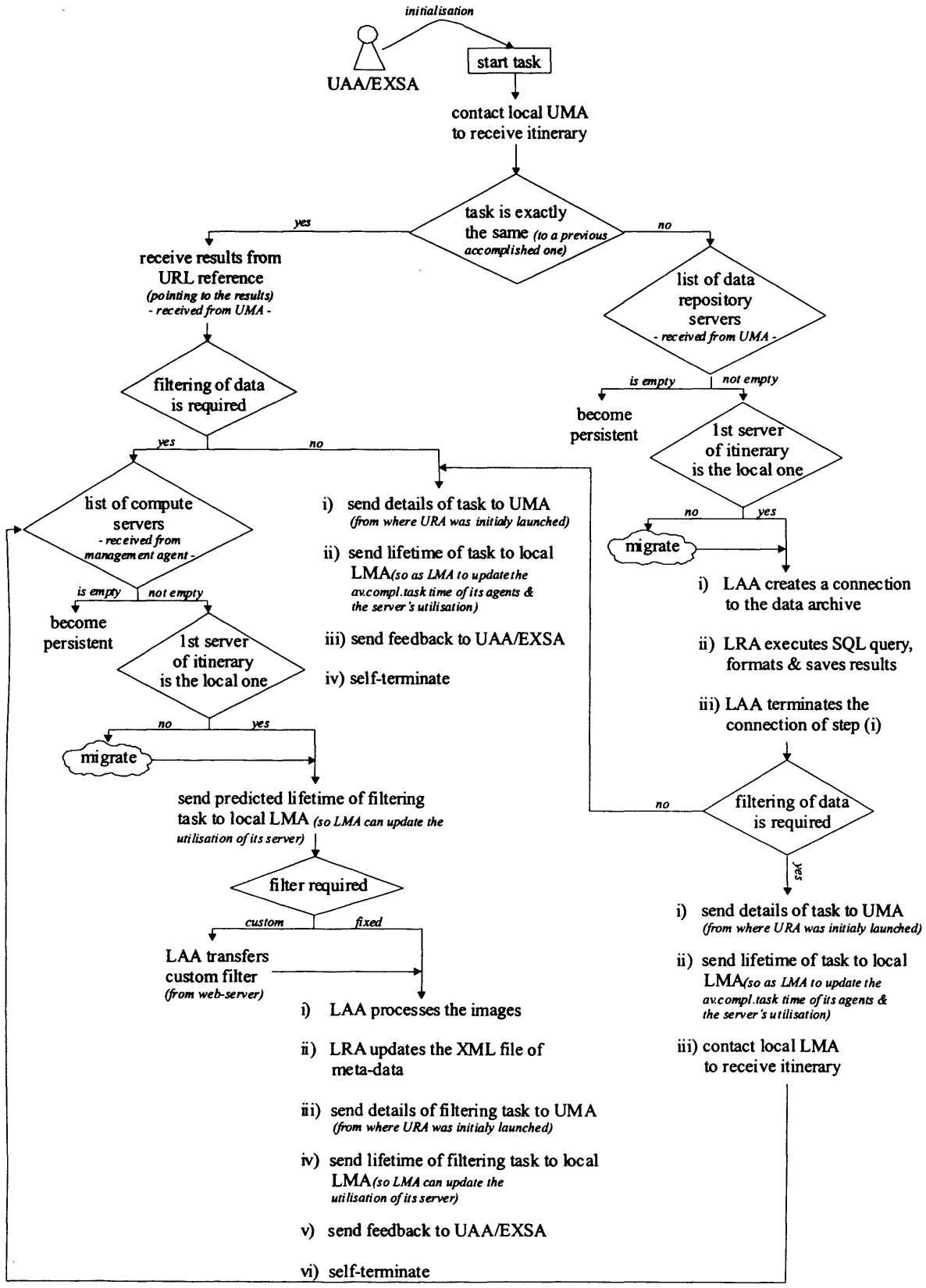


Figure 7.7. The basic algorithm of URA

7.2.2.3. UMA - Universal Management Agent

A UMA is similar to an LMA in that it creates the itinerary of a URA, balancing at the same time the load of mobile agents among the information-servers. The functionality of UMA is like LMA's with three extra *methods* to:

i) check if a mobile agent's task is similar to a task accomplished by another agent in the past,
ii) receive from a URA details on the progress of its task,
iii) exchange information on URA's personal details with the other UMAs via their private sub-SPACE using multicast messaging.

The only difference between a UMA and an LMA is that the former only receives the system status information.

7.3. Implementation considerations

The development of agents in Java programming language and the existence of configuration files that separate server-depended features from agents' code produce agents that are fully portable to run on any server. As a result every information-server or web-server has a replicated collection of the same agents regardless of the platform, operating system, storage medium, data repository or computation facilities employed.

A server may be dynamically added or removed to/from the system by including or excluding the corresponding server's management agent to/from the list of available management agents in the *man_agents.inf* configuration file of every server. The introduction of a new server however requires a server-depended configuration file to reflect its specifications, but not any modifications to the code of its agents.

In the same sense, the modularity of every agent enables the easy modification of a specific Class without affecting the rest of the agent's code, and provides the flexibility of attaching a new Class to an agent as an extension to the services provided by it. The multithreaded nature of agents allows the concurrent execution of different tasks from multiple users simultaneously, and the ability of users to

provide their own analysis algorithms. This allows the fusion of results of their queries, and contributes to the extensibility and customisability of the system.

7.4. Conclusion

Apart from the complexity of the code for implementing the prototype, the magnitude of the prototype system and the time-consuming validation procedures required to ensure its proper operation, a lot of difficulties have been encountered during the implementation. Most of them regard bugs in the software applications used for the development of the prototype. Although Voyager and FIPA-OS are continuously improved and new versions are being released, they both lack proper documentation. For instance, the code-examples of the official Voyager documentation on mobility are referred to as agents migrating to different agent platforms hosted by a single machine instead of remote ones, where the syntax of commands differs. In contrast, the forums and mailing lists of Voyager and FIPA-OS offer a vital source of information. Part of SARA prototype has been supplied to University of Edinburgh, school of Mathematical and Computer Sciences[17] for use in a research project supervised by Dr. Phil Trinder that attempts to discover mobility design patterns for the realisation of an open source library of generic abstract Java Classes. Finally, experimental tests on the interoperability of the architecture have been conducted in collaboration with Dr. Anthony Karageorgos from the department of Computation in University of Manchester Institute of Science and Technology (UMIST) [2].

Chapter 8. Experiments and Results

8.1. Introduction

This chapter contains screen-shots, code-results, charts and statistics of the experiments conducted on the prototype of SARA active digital library. The results demonstrate the successful achievement of System Integration and Data Management within the agent-based architecture of collaborative agents, the even utilisation of the information-servers on balancing the agent load, and the optimisation in performance provided by the adaptability of the LB model. The system's ability on interoperating with external FIPA-compliant agent-based systems is also demonstrated.

8.2. Accessing SARA active Digital Library from the web

Access to SARA active Digital Library from the web is achieved through a GUI depicted in Figure 8.1. The initial HTML-page of the SARA web-site enables a user to perform a request on a collection of SAR images specified by an SQL query. The resulting image can be further analysed using an image processing algorithm.

The top frame of the HTML-page is used as a form where a user enters the information required for the accomplishment of his/her request. This involves his/her user-name and password, an SQL query, and the optional usage of an image processing algorithm to filter the data. A user may choose one of the four fixed filters to filter the data or provide his/her own analysis algorithm i.e. custom filter. The bottom frame is used for the visualisation of results.

The user does not need to have any knowledge about the underlying infrastructure i.e. which information-servers need to be accessed or the availability of computational resources employed by each of them. Once an agent is launched from a user, the user is free to do other tasks. When the agent accomplishes its objective it returns to its user a URL reference pointing to the results of the request, see bottom frame of HTML-page. Actually, the URL address reveals the structure of a user's file-space. For instance the user-name, the request's execution date and time, the web-server from where the agent was initially launched etc. form the directory-names of the corresponding user's file-space structure. An example of data results pointed to by such a URL reference can be found in Code 8.4a of *section 8.3*.

An alternative approach of providing a user with the results of his/her request would be by e-mailing the appropriate URL reference (pointing to the actual results) to him/her. This is extremely useful in highly time-demanding tasks where a user does not need to have a permanent network connection until his/her request is fulfilled. This could be easily achieved by programming the URA to email the URL reference to the appropriate user in the event where his/her corresponding UAA agent is terminated i.e. the HTML-page from where the user launched the agent has been closed.

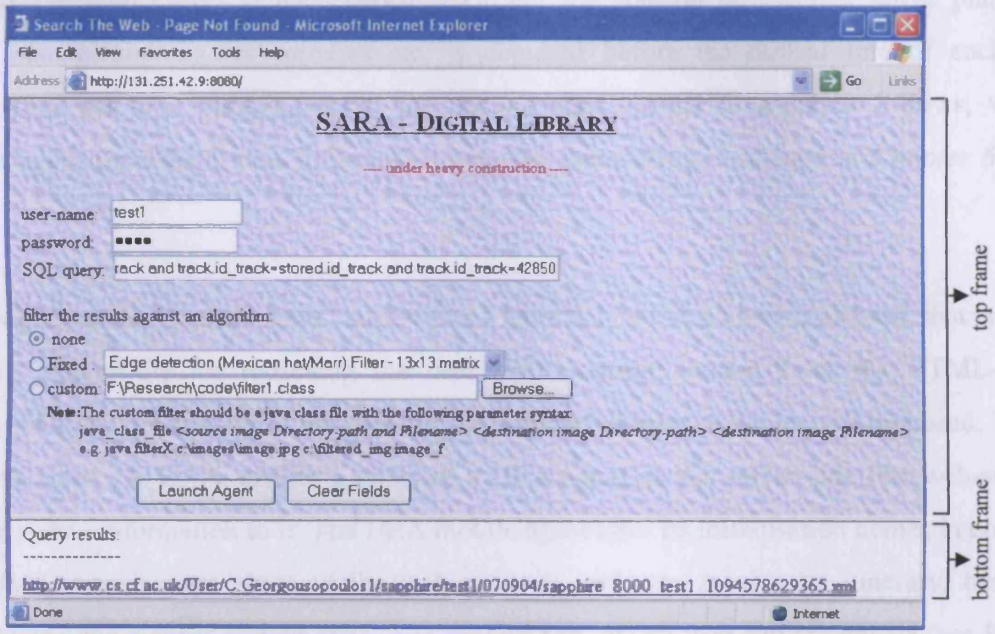


Figure 8.1. The SARA initial web-page

The prototype developed does not inform the user of the process time required for the accomplishment of his/her request. Estimation on the total completion time of a task before its initiation is a difficult procedure, and sometimes impossible. For instance, the exact time of an SQL query execution cannot be calculated beforehand, though a prediction on the time needed to process the data gathered by such a query against an image processing algorithm is feasible; based on the amount of data and the server's throughput on where that data will be processed. A possible approach of providing the user with details on the progress of his/her task is by monitoring the status and location of URA during its execution, and is part of the feature work as mentioned in *Chapter 9 - section 9.2.*

8.2.1. Procedure of accomplishing a request

The accomplishment of a user request consisting of a single SQL query, an SQL query with a fixed filter, and an SQL query with a custom filter performed from the SARA web-based interface is illustrated in Figures 8.2 and 8.3. Figure 8.2 corresponds to a screen-shot of the web-server console and 8.3 to a screen-shot of two information-server consoles accordingly.

The initialisation information of each server regarding the start-up of Voyager agent platform, the creation of management agents' proxies etc. is depicted before the dashed line of each console separately. After that the initialisation of the management agents' information follows, where the management agents exchange their information between themselves; discussed in *Chapter 6 - section 6.3.1.4.*

When a user presses the "launch agent" button (see Figure 8.1) a UAA agent is created, that is in a form of a servlet, responsible for retrieving the user's information entered from the HTML-page and launching a URA agent on behalf of him/her, for the accomplishment of the request proposed. The user's representative agent i.e. UAA, creates a proxy of a URA agent on the server-side (the web-server) and forwards the user's information to it. The URA mobile agent after its initialisation comes in contact with the local UMA management agent of the web-server in order to receive its itinerary, based on its requirements and the current system state. The second part of the web-server console, see Figure 8.2, demonstrates the UMA's response to URA's request for itinerary. UMA forms a list of servers with the available resources identified by their IP addresses along with their utilisation values that can serve the agent's request. URA's itinerary is then constructed (in ascending order) of servers according to their utilisation values.

Upon receipt of an itinerary, the URA migrates to the first server placed on the list. The progress of URA on the information-server to which it migrates is displayed in Figure 8.3. In fact, the figure demonstrates the execution of different URA agents in accordance with the three user requests. Each URA's steps of execution are pointed out by an arrow of different colour. URAs can also be distinguished based on their names enclosed in brackets; especially by looking at the number at the end of each mobile agent's name.

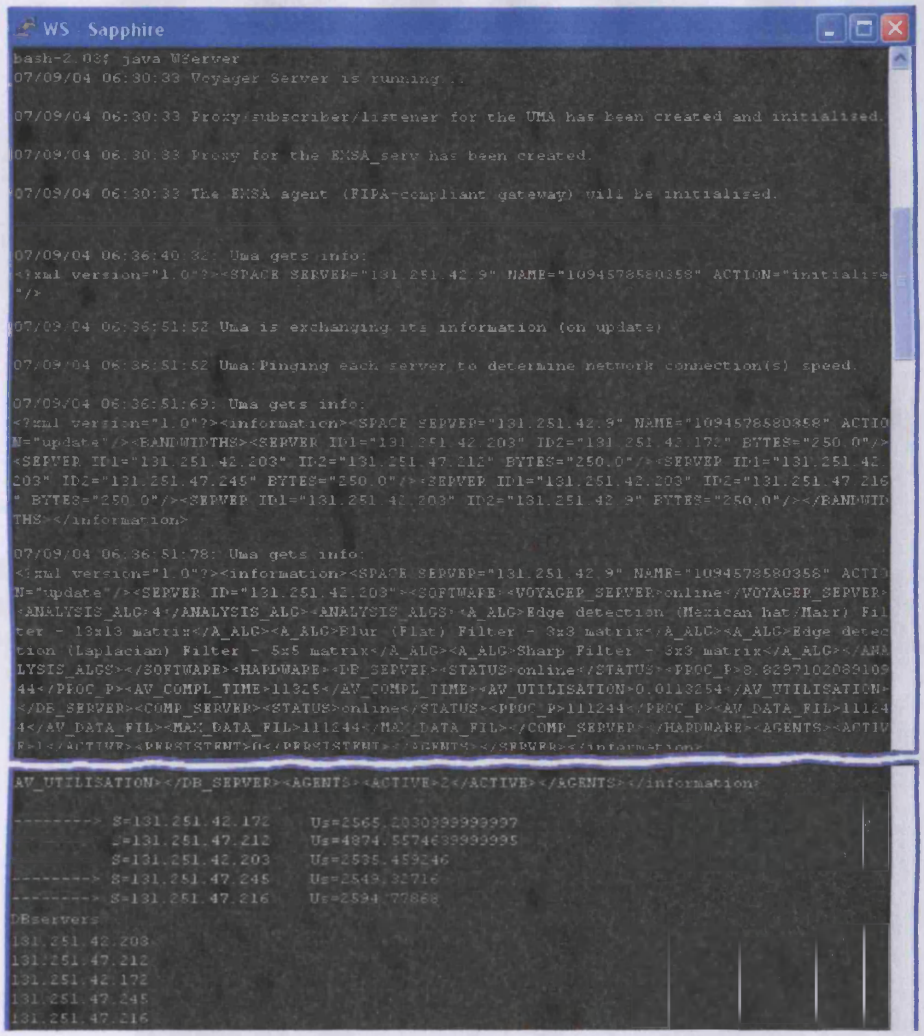


Figure 8.2. The web-server console

The agent activities are also illustrated in a sequence diagram in Figure 8.4. The steps of URAs execution pointed out by different colors in Figure 8.3 are depicted in Figure 8.4 as well; in bold letters enclosed in brackets.

Initially, after it has successfully migrated to *Bloodstone* information-server (step 1) each URA comes into contact with the LAA agent. LAA's responsibility is to provide the incoming URAs with information on how to access the server's data repository, since the information source is wrapped by LAA stationary agent. The data repository of each information-server in the SARA prototype is

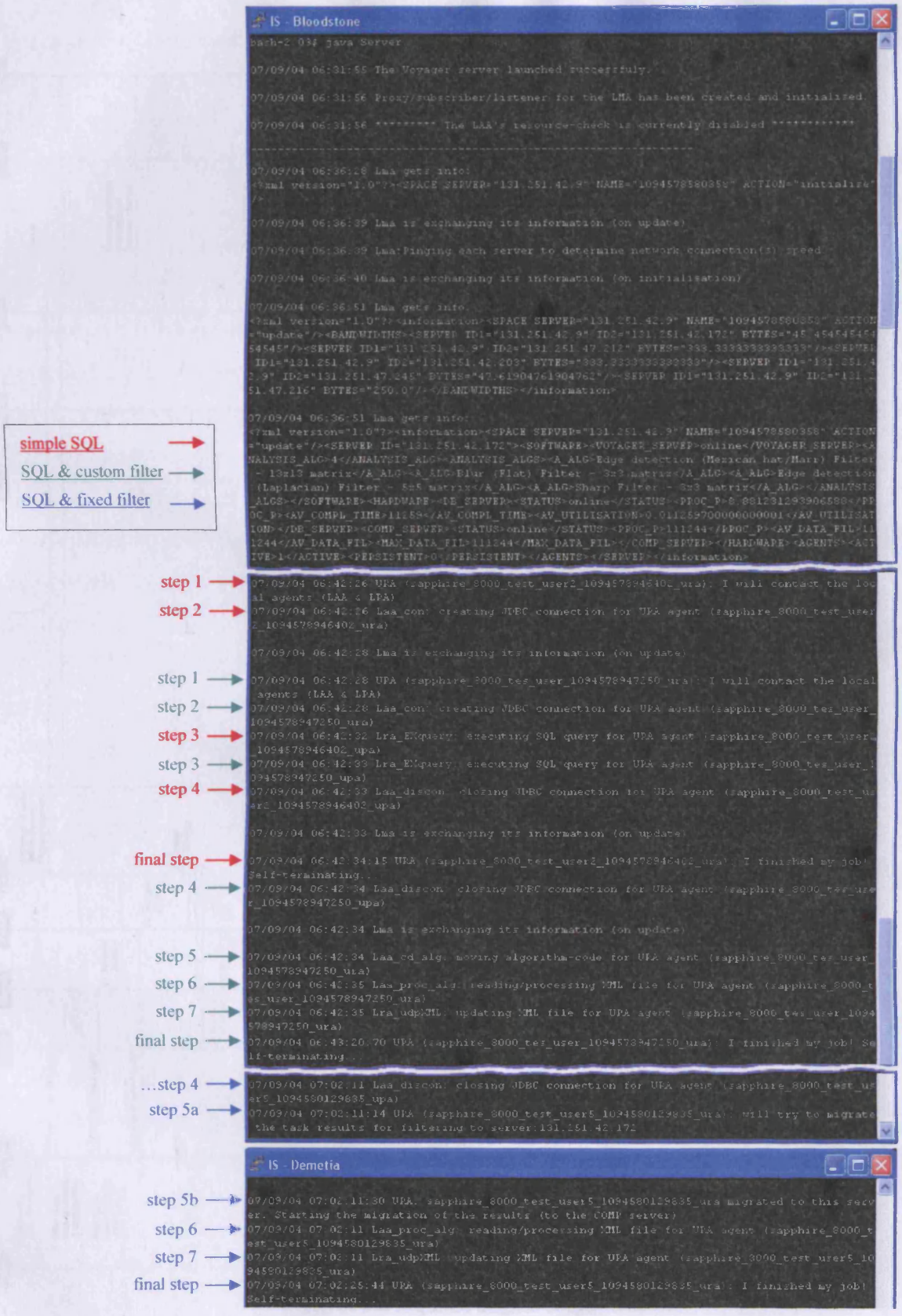


Figure 8.3. Two information-server consoles

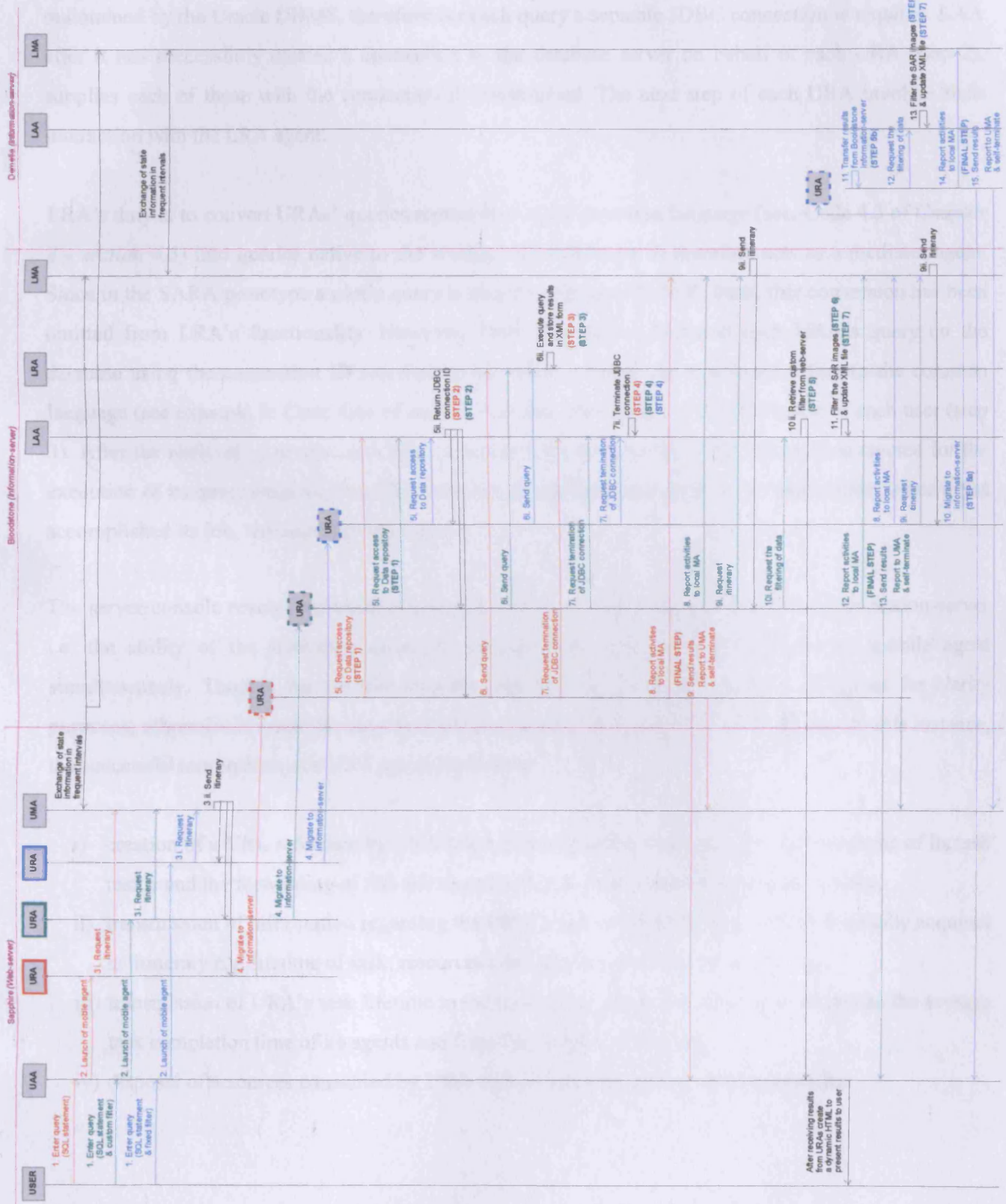


Figure 8.4. Sequence diagram of agent activities

maintained by the Oracle DBMS, therefore for each query a separate JDBC connection is required. LAA after it has successfully created a connection to the database server on behalf of each URA (step 2), supplies each of those with the connection ID established. The next step of each URA involves their interaction with the LRA agent.

LRA's duty is to convert URAs' queries represented in the common language (see, Code 4.3 of *Chapter 4 - section 4.3*) into queries native to the sources and vice-versa. It therefore acts as a mediator agent. Since in the SARA prototype a user's query is directly expressed in SQL form, this conversion has been omitted from LRA's functionality. However, LRA after it has executed each URA's query on the database using the connection ID supplied by the latter, formats the results according to the common language (see example in Code 8.4a of *section 8.3*) and stores them in the file-space of each user (step 3). After the retrieval of results each URA instructs LAA to close the JDBC connection created for the execution of its query (step 4). The URA with the simple task (identified by the red colour), since it has accomplished its job, terminated (final step).

The server console reveals the multi-threaded execution of mobile agents within the information-server i.e. the ability of the stationary agents to provide their services to more than one mobile agent simultaneously. Though, the console displays only the most important actions of agents for clarity purposes; otherwise it would be extremely difficult to trace the operation of each agent. In this instance, the successful termination of a URA agent implies the:

i) creation of a URL reference by URA itself, pointing to the location of where the results of its task reside and the forwarding of this address to its UAA creator for visualising the results,
ii) transmission of information regarding the URA's task to the UMA from which it initially acquired its itinerary e.g. lifetime of task, resources used, physical location of results etc.,
iii) transmission of URA's task lifetime to the local LMA, in order for the latter to update the average task completion time of its agents and form the server's utilization,
iv) disposal of resources consumed by URA such as memory, after its self-termination.

After each URA's termination its UAA creator, on receipt of a URL address, posts it to the html-page of the corresponding user.

The execution of the URAs with complex tasks, identified by the green and blue colour, continues after they have both received their itinerary composed of a collection of compute servers from the local LMA, to proceed with the filtering of data collected in steps 1-4. The green URA continues the second part of its task i.e. the filtering, on the same information-server from which it gathered the results of its query, whereas the blue one migrated to another compute server (step 5a,b) to perform its filtering. This was either due to unavailability of compute resources (i.e. the specified fixed filter did not exist on *Bloodstone*) or because the blue agent's filtering task was predicted by LMA to be accomplished fastest on *Demetia*.

In simple agent tasks the utilisation of a server is updated by the corresponding LMA on the arrival and departure (or self-termination) of URA agents. This is done by adding or subtracting an agent to/from the current agent load, based on Malone's formula of utilisation: $(a \cdot \mu / L)$. In complex agent tasks the predicted lifetime of a URA's filtering task (calculated by the local LMA/UMA of which URA receives its itinerary) is added as extra time to the agent load of the utilisation of the server on which URA will migrate to perform its filtering task. The URA on its arrival to the visiting server provides the local LMA with the predicted time of its filtering task so as the server's utilisation is being updated. After the filtering task is completed, URA returns to the local LMA the exact processing time required; in continuation LMA forms the utilisation of the server by subtracting that time from the current agent load.

The custom analysis algorithm of a user supplied with a query is initially maintained in the web-server on where the user posts his/her request. After the user's query results have been gathered, the custom analysis algorithm has to be transferred from the web-server to the information-server on where the filtering procedure will commence. Therefore, the execution of the green URA's filtering task requires the transfer of a user's custom analysis algorithm from the web-server (*Sapphire*) to the information-server (*Bloodstone*), on where URA currently resides ready to perform its filtering. LAA's action of transferring the custom filter on behalf of green URA is identified in step 5. Step 5a of blue URA

indicates its intension of migrating to *Demetia*, whereas step 5b illustrates the successful URA's migration to the new information-server, followed by the results migration from *Bloodstone* to *Demetia*.

The actual filtering of data held by URAs is conducted in steps 6 and 7. LAA (step 6) accesses the XML file containing the metadata that describe each mobile agent's image collection (data) and processes each image in a separate *thread* against the filter specified by each URA. Concurrently, LRA (step 7) constructs a new XML file including the metadata for the images generated after the processing of the original ones. An example of an XML document before and after its update that contains information for the new resulting images can be found in Code 8.4 of *section 8.3*; the extra information is shown in bold pointed out by the arrows.

The successful termination of URAs after their filtering implies the creation and forwarding of a URL address to the UAA, the transmission of complementary information regarding their filtering task to the appropriate UMA, and receipt of the filtering task lifetime by the local LMA to form the server utilisation.

Finally, notice that every stationary agent consists of different Java Classes e.g. LAA is comprised of *Laa_con*, *Laa_discon*, Laa_*cd_alg*, *Laa_proc_alg*, and *Laa_rc* (LAA's *r*esource *c*heck on users' file-space, not enabled in the above experiment). The separation of a stationary agent's functionality into different parts enriches the modularity and extensibility of the agent. In this instance, a change or addittion of a new service provided by an existing agent can be easily achieved. For example, the maintenance of an information-server's data repository by a future DBMS may require updates only to the *Laa_con* and *Laa_discon* parts of LAA.

8.2.2. Load balance within the agent-based architecture of SARA

The experiments conducted on load balancing were performed on a 100Mbit/s Fast Ethernet network with six Sun Ultra 5 Workstations of a 270 MHz UltraSPARC-IIi 64-bit processor running Solaris 8 operating system, utilising the Voyager agent platform. Five of them were used as information-servers and one as a web-server. Every information-server had a data repository maintained by the Oracle DBMS, composed of replicated test-data (see, *Appendix A1*) with identical computational facilities.

Figure 8.5 is a screen-shot taken during the operation of five information-servers and a web-server remotely logged from a computer running Windows XP. The *Sapphire* web-server console is depicted on the top left corner, the *Illinois* information-server on the bottom left corner and the rest of the information-servers (*Bloodstone*, *Kentucky*, *Lassus*, *Demetia*) on the right side. The creation of the management agents' SPACE necessary for information exchange between them is constructed by the manual execution of *setupSP* program after the initialisation of each server, depicted in the middle console window on the left side of Figure 8.5

After the creation of the SPACE, a message is sent to every management agent of each server with the SPACE details[1] and the instruction (*action*) of information initialisation; see Figure 8.2 or 8.3 after the dashed line. In response, every management exchange its local system state information using the SPACE; the transmit of this information is also used to determine the initial network bandwidth between servers, since the messages exchanged are time-stamped. In addition, every management agent on initialisation (afterwards, on regular basis) *pings* the rest of the management agents to determine the network latency between them. For instance, at the bottom of the first window in Figure 8.2 is demonstrated the receipt of *Bloodstone* server's system state information (IP address: 131.251.42.203) from *Sapphire* web-server, and in Figure 8.3 the receipt of *Demetia* server's system state information (IP address: 131.251.42.172) from *Bloodstone* information-server.

Further exchange of management agents' updated information during mobile agent execution may be observed in the console windows of each server in Figures 8.2, 8.3 and 8.5. For example, a change in utilization of *Kentucky* information-server (IP address: 131.251.47.216), is posted to every management agent registered to SPACE; pointed out by the red arrows in Figure 8.5.

Figure 8.5 also demonstrates the actions of a URA which attempts to migrate to a failed information-server. During the launching of URA agents, the *Demetia* information-server (on the bottom right corner of figure) has been shut-down; pointed out by the yellow arrow. The URA which has been assigned to *Demetia* and was the first agent to migrate to that server after the server's failure, pointed out by the green arrow, on its attempt it determines that the server's agent platform was shut-down.

[1] *The name of a SPACE and the IP address of the server hosting it.*

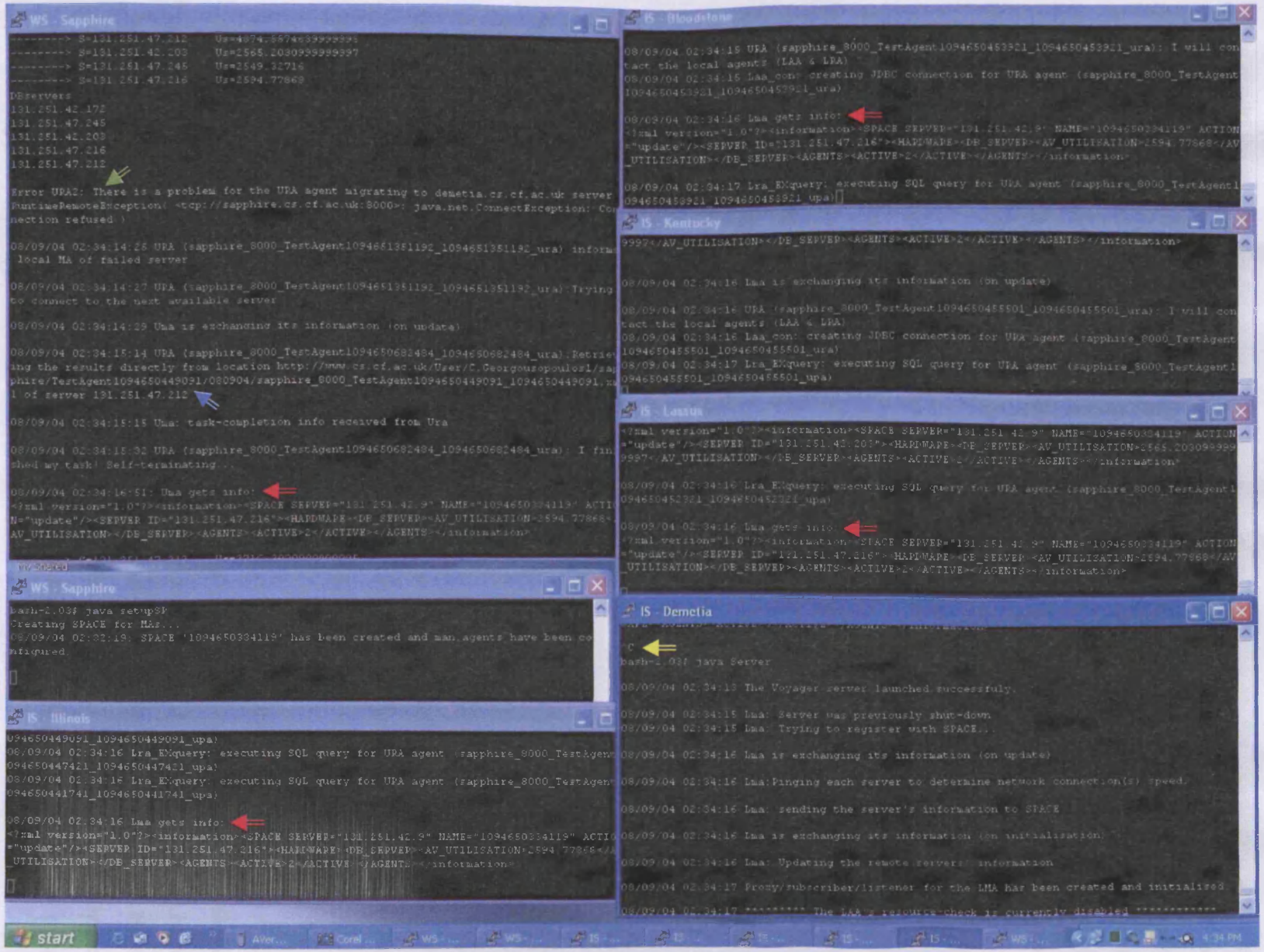


Figure 8.5. 5 information-servers and 1 web-server in operation

As a result, the agent informs its local management agent i.e. UMA, of the failed server and tries to migrate to the next server in its itinerary. Subsequently, the UMA posts this information to SPACE in order for every management agent to be aware of the failed server. In addition, the blue arrow on the web-server console indicates a URA which has been instructed by UMA to retrieve the results of its task directly from the location stored by another agent in the past, due to a similarity of agent tasks. The mobile agents task required only the creation of a URL reference pointing to the results, and the forwarding of this to its UAA.

After *Demetia* was initialised for a second time, its management agent identified that the server was previously down. Then LMA searches for an *alive* management agent, receives the SPACE details from it via direct messaging and registeres itself to SPACE. LMA on its initialisation *pings* the other the management agents and posts its system state information to SPACE. Finally, it acquires the system state information for the rest of the servers from the management agent from which it had initially received the SPACE details. *Demetia* is then ready to server any incoming URAs.

Experiments on balancing the load of mobile agent tasks among the information-servers yielded positive results on the even utilisation of every server, each server shown similar levels of utilisation. The agent tasks used on the experiments were both simple and complex. A simple task involved the information acquisition of one or more SAR images, whereas a complex one the information acquisition of a single SAR image and its image processing against a fixed filter. The time for an agent to be serialised and migrate to an information-server was 595-725 milliseconds (ms) and its execution on server-side required 1,242-1,712 ms, resulting it a total time of 1837-2437 ms for a simple agent task to be accomplished. The time required for an image to be processed was 10,863-11,135 ms.

The chart in Figure 8.6 displays the utilisation of each of the five information-servers used in the SARA prototype during the launch of 200 agents with simple tasks, and demonstrates the even distribution of agent load among the servers. Note that the utilisation of a server in SARA LB model does not represent its actual CPU usage but its agent load (the expected time of when a server will be unloaded). Since the utilisation of a server is updated after the arrival and before the departure of an agent from that server, small rises and drops on the graph of every server are expected to appear due to the intervals of sampling

values recorded. The fact that the utilisation of all servers in every single point of time during the launch of agents is on a similar level implies that the servers will be unloaded equally. This proves that the task assignment policy followed, distributed the agent load between the available servers in such a way that all servers will be utilised the same.

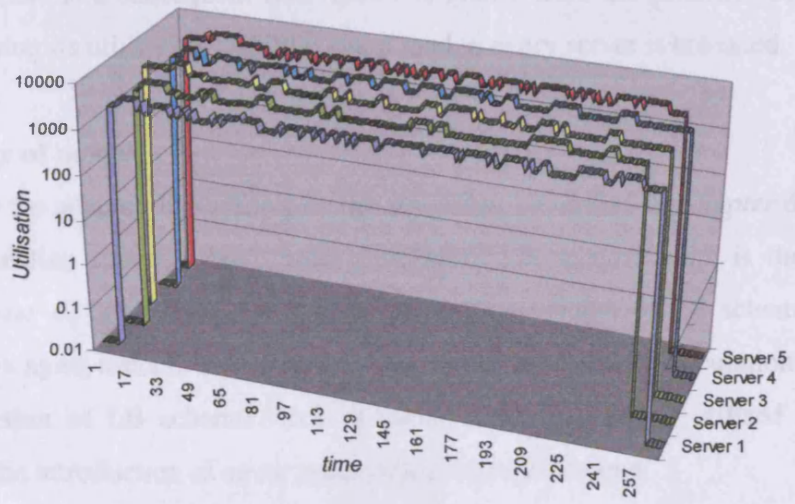


Figure 8.6. Representation of information-servers' utilisation on execution of simple agent tasks

The introduction of agents with complex tasks in the agent load resulted in higher deviations of a server's utilisation. The chart in Figure 8.7 illustrates the utilisation of the information-servers on which 15% of the agents launched had complex tasks.

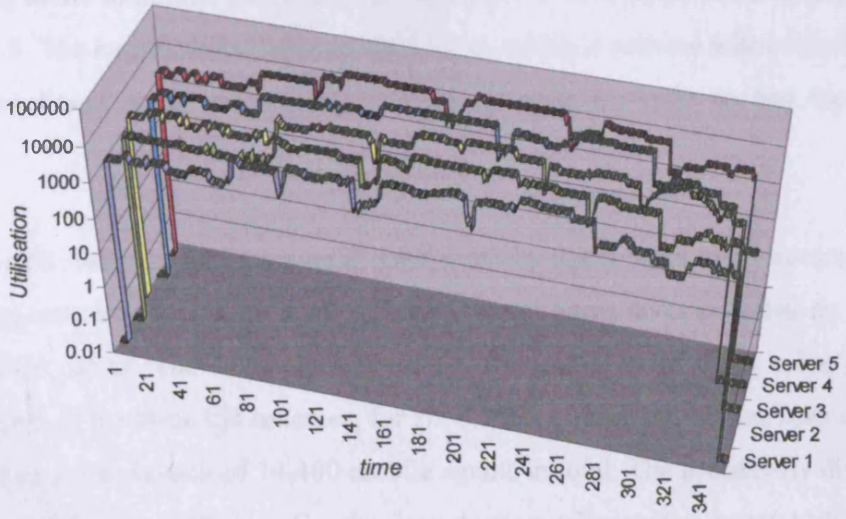


Figure 8.7. Representation of information-servers' utilisation on execution of mixed agent tasks

Variation in the graph of each server are higher in comparison with the previous chart, due to the arrival/departure of agents with complex tasks that require more time to be accomplished than simple ones. Though it can be observed that the utilisation of each server fluctuates in the same way to the rest of the servers, where after a high drop in the graph of a server caused by the completion of one or more complex task(s), there is a subsequent rise. This rise comes from the constant assignment of agents to that server, increasing its utilisation until the agent load in every server is balanced.

8.2.3. Adaptability of model

In order to explore the adaptability offered by the algorithm, described in *Chapter 6 - section 6.4.1*, three different load balancing schemes have been developed. LB scheme No.1 is the SARA LB scheme discussed in *Chapter 6*. LB scheme No.2 is an alternative version of LB scheme No.1 in which the lifetime of complex agent tasks is unknown and therefore is not used in calculations. LB scheme No.3 is an alternative version of LB scheme No.2 in which our algorithm is utilised for amending server utilisation due to the introduction of agent tasks with unknown lifetime.

Through experimentation it was determined at what percentage LB scheme No.3 reaches the performance of LB scheme No.1. This was done to test the functionality of the algorithm utilised by a system where the lifetime of complex tasks cannot be estimated or predicted successfully. The performance of each load balancing scheme on distributing 200 agent tasks among five information-servers, according to the total time required by those agents to accomplish their tasks, is presented in the chart of Figure 8.8. The experimental tests performed on each LB scheme within the SARA agent-based system have been based on a variable introduction of complex tasks to test the efficiency of the algorithm.

The less the rate of a series is, the less overall time spent by agent tasks to be accomplished. Therefore, the load balancing scheme which results to the least time of agent tasks completion, corresponds to the best task assignment policy. The value of each series corresponds to the mean value obtained from four experiments on each of the three LB schemes, for six different variable introductions of complex tasks in agent load, resulting in the launch of 14,400 mobile agents in total. The probability distribution followed was an agent launch every 1500 ms. On the introduction of complex agent tasks, an agent with a

complex task was launching after every three agents of simple tasks. According to the complex tasks increase (from 0% to 25%) in agent load, a delay of 3000-9000ms was necessary to be introduced after the launch of every 40 agents, in order not to overload the servers. When servers where overloading there was a network lag on management agents' communication that could be only observed at the end of the experiment, where even though all agents have finished their tasks the management agents were exchanging information on agent migrations. The exact point at when a server overloads is discussed in *Chapter 9 - section 9.4* as part of the future work associated with this research.

LB scheme No.1, which is based on known or correctly predicted lifetime of agent tasks, disseminates properly the agent load among the servers by utilising evenly each server and therefore resulting in the fastest completion of agent tasks in comparison with the other two LB schemes. When there are no complex tasks involved all of the three LB schemes behave the same.

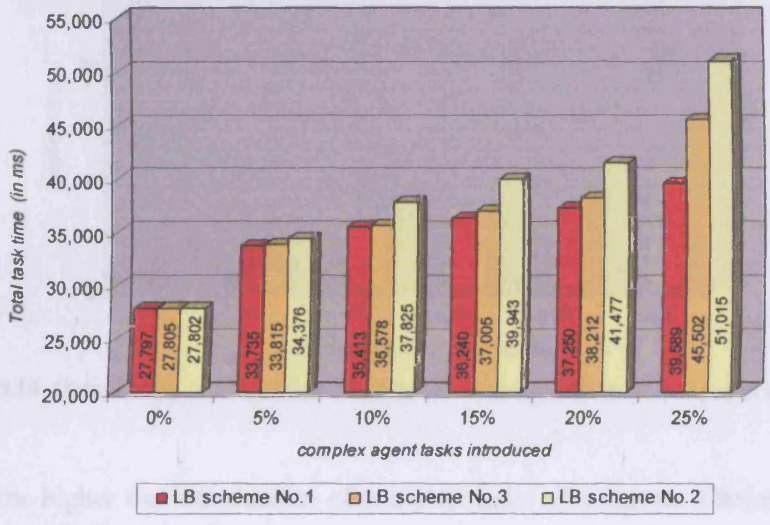


Figure 8.8. Total task time required by agents to complete their tasks

The difference in load balancing performance between LB scheme No.1, No.2 and No.3 is expressed in the chart of Figure 8.9. The chart illustrates by what percentage LB scheme No.1 is better than scheme No.2 and No.3. Difference in performance between schemes No.2 and No.3 is due to the utilisation of our algorithm as shown in the chart of Figure 8.10.

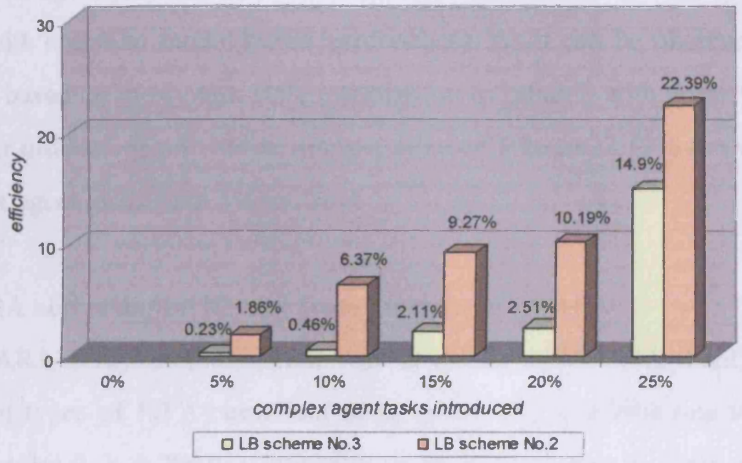


Figure 8.9. LB scheme No.1 versus No.2 and No.3

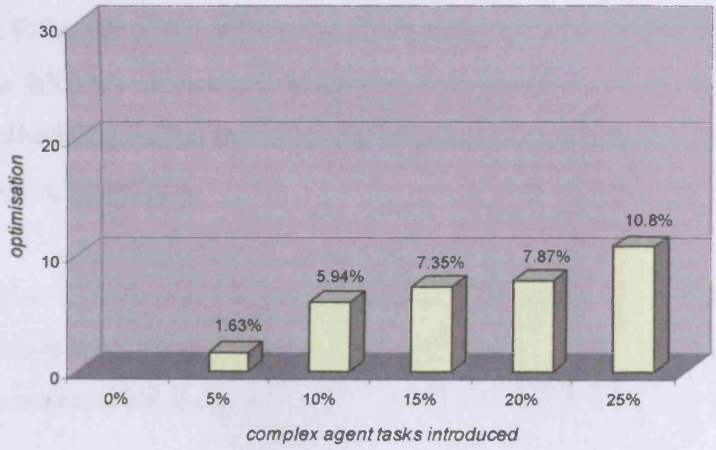


Figure 8.10. Optimisation of LB scheme No.2, based on the utilisation of the special algorithm

It can be seen that the higher the introduction of complex tasks of unknown lifetime in a system (from 5% to 25%), the better the load balancing achieved by the utilisation of our algorithm (with an improvement of 1.63% to 10.8%).

In addition, Figure 8.9 demonstrates the advantage of the proposed load balancing scheme that is based on the combination of state-base and model-based approaches of LB - represented by LB scheme No.1, over the LB scheme No.2 that is based only on system state information (as common state-based approaches do). This is because, the lifetime of complex agent tasks is unknown in LB scheme No.2, and

any predictions on the agent task lifetimes, and therefore estimations on server utilisations cannot take place (in contrast with common model-based approaches). As it can be observed from the chart, load balancing decisions based on the system state information in relation with predictions on the lifetime of agent task and server utilisations provide an improvement of 1.86 to 22.39% (on variable introduction of agents with complex agent tasks from 5% to 25%).

8.3. Accessing SARA active digital library from an external MAS

The ability of the SARA system to interoperate with an external FIPA-compliant system has been tested against two different types of FIPA-compliant agent platforms. The first one was implemented using FIPA-OS toolkit (version 2_1_0-20030219000011, build:314) running on Unix and the second one was implemented on JADE toolkit[85] (version 2.4.1) running on Linux.

The test agent of the FIPA-OS agent platform was created to search the Directory Facilitator (DF) of SARA system for the EXSA's service and perform a Request. The second test agent was developed using the JADE agent building toolkit located at the Manchester Agentcities[1] node, which is hosted at the Dept. of Computation, UMIST[2].

The top console window of Figure 8.11 is a screenshot of the SARA web-server console (running on Windows XP), the middle one is the console of the SARA information-server (running on Unix) and the last one shows the execution of the test agent using the FIPA-OS agent platform (running on Unix).

Initially, both of the test agents perform a search on the DF of SARA to find the EXSA gateway agent AID (Agent IDentifier). The interaction of an agent with the SARA DF is managed by FIPA-OS itself, using which FIPA-compliant gateways have been implemented. Once, the test agents have acquired the gateway agent AID, they both send a Request performative to EXSA, similar to the following:

```
(request
:sender agent_from_EX MAS_id
:receiver EXSA_id
:content (<?xml version="1.0" ?><ex_SARA_mes>
          <coordinates c1="16.317"  c2="107.654" c3="16.061"
                       c4="108.082" c5="16.828"  c6="108.575"
                       c7="17.087"  c8="108.144"/></ex_SARA_mes>)
:language XML
:ontology EX_SARA_ontology.dtd
...)
```

Code 8.1. Example of a simple Request ACL message

This is constructed based on the ontology of the service provided by EXSA depicted in Code 5.3 of *Chapter 5*.

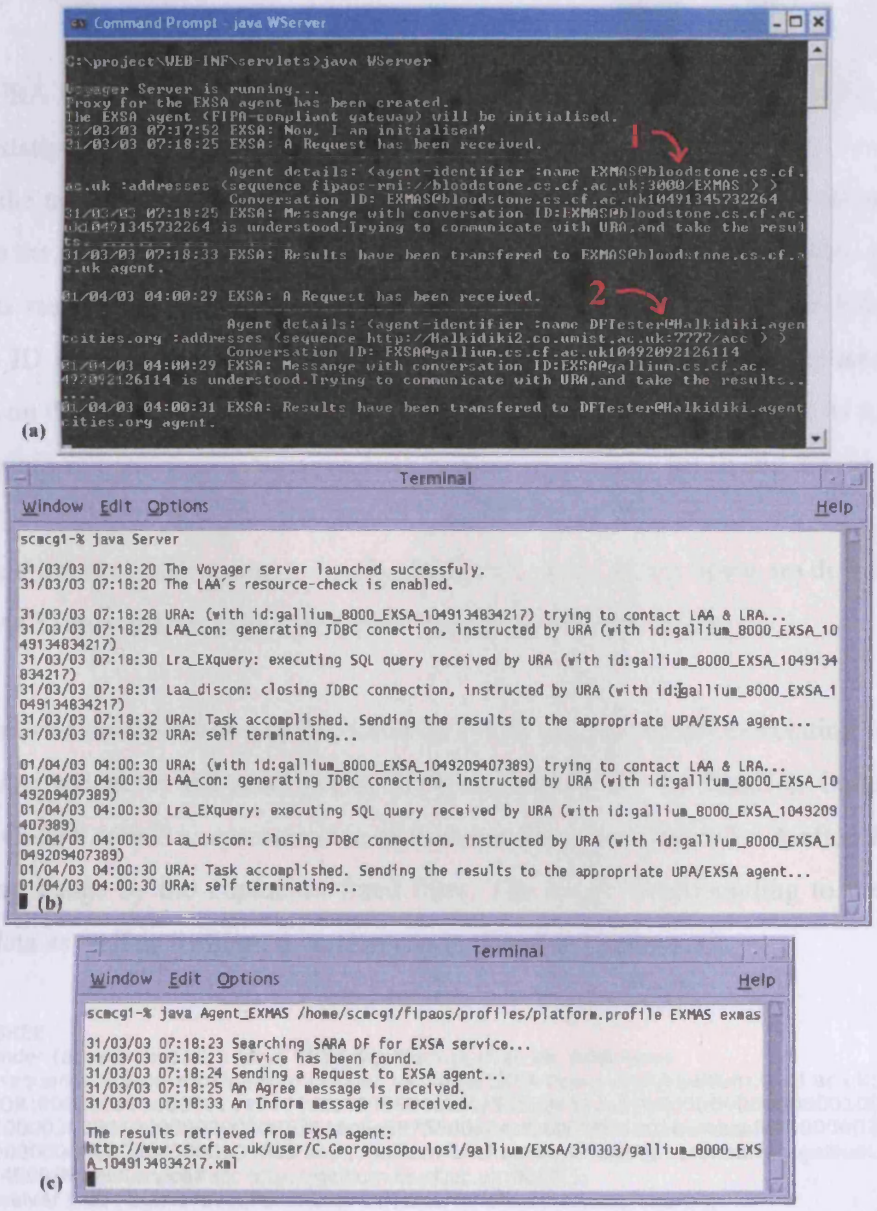


Figure 8.11. Server consoles

The coordinates specified in the content of the ACL message correspond to the query the sender agent. When EXSA receives the requests from the test agents (Figure 8.11a) it validated them, and replies to each of the test agents with an Agree performative (Figure 8.11c). A URA agent is created locally for

each request. The content of the messages sent to each URA from EXSA are: the test agents' request translated into the form understood by URA (the XML content of the message) and the conversation ID of the corresponding test agent interaction with EXSA.

After each URA has been initialised by EXSA, it migrates to the information-server, and interacts with the server's stationary agents (Figure 8.11b) in the same manner as described in *section* 8.*2.1*, in order to accomplish the task assigned by EXSA. In the experiments a single information-server has been used and therefore the interaction of URA with the management agents has been ignored. After each URA has completed its task, it sends a URL reference pointing to the results back to EXSA along with the conversation ID (initially received by EXSA) and terminates. Then EXSA replies to each of the test agents based on the conversation indicated by the conversation ID received from its internal agent i.e. the URA, via an Inform performative with the URL address (see Figure 8.11b and 8.11c).

The Agree and Inform ACL messages sent by the EXSA to JADE test agent are depicted in Code 8.2 and 8.3; similar ACL messages are also received by the FIPA-OS tester agent.

The data retrieved based on the coordinates specified by the test agents in relation to the test data (see, *Appendix A1*) provided by the prototype of SARA active DL, can be found in Code 8.4(a), encoded in XML form. Code 8.4(b) is a representation of the same data modified by LAA after the processing of the corresponding image by the *Laplacian* fixed filter. The image corresponding to the description of the above metadata as well as its filtered version can be found in Figure 8.12.

```
(AGREE
:sender (agent-identifier :name EXSA@gallium.cs.cf.ac.uk :addresses
  (sequence fipaos-rmi://gallium.cs.cf.ac.uk:3000/EXSA fipaos-rmi://gallium.cs.cf.ac.uk:3000/acc
  IOR:000000000000001149444c3a464950412f4d54533a312e3000000000000000001000000000
  000003000010000000000000867616c6c69756d0004cf000000000018afabcafe000000026dc432d3
  0000000800000000000000000 iiop://gallium.cs.cf.ac.uk:4000/acc corbaname::gallium.cs.cf.ac.uk
  :4000/NameService#acc http://gallium.cs.cf.ac.uk:8080 ))
:receiver (set ( agent-identifier :name DFTester@Halkidiki.agentcities.org
:addresses (sequence http://Halkidiki2.co.umist.ac.uk:7777/acc )) )
:content "(<?xml version="1.0" ?><ex_SARA_mes><coordinates c1="16.317" c2="107.654"
   c3="16.061" c4="108.082" c5="16.828"  c6="108.575" c7="17.087" c8="108.144"/>
   </ex_SARA_mes>)"
:language XML
:ontology EX_SARA_ontology.dtd
:conversation-id EXSA@gallium.cs.cf.ac.uk104920921126114
)
```

Code 8.2. Agree ACL message received from the JADE tester agent

```
(INFORM
:sender ( agent-identifier :name EXSA@gallium.cs.cf.ac.uk :addresses
  (sequence fipaos-rmi://gallium.cs.cf.ac.uk:3000/EXSA fipaos-rmi://gallium.cs.cf.ac.uk:3000/acc
  IOR:000000000000001149444c3a464950412f4d54533a312e3000000000000000001000000000
  0000030000100000000000867616c6c69756d0004cf000000000018afabcafe000000026dc432d3
  000000080000000000000000 iiop://gallium.cs.cf.ac.uk:4000/acc corbaname::gallium.cs.cf.ac.uk
  :4000/NameService#acc http://gallium.cs.cf.ac.uk:8080 ))
:receiver (set ( agent-identifier :name DFTester@Halkidiki.agentcities.org
:addresses (sequence http://Halkidiki2.co.umist.ac.uk:7777/acc )) )
:content "http://www.cs.cf.ac.uk/user/C.Georgousopoulos1/gallium/EXSA/010403/gallium
  _8000_EXSA_1049209212627.xml"
:language ASCII
:conversation-id EXSA@gallium.cs.cf.ac.uk104920921126114
 )
```

Code 8.3. Inform ACL message received from the JADE tester agent

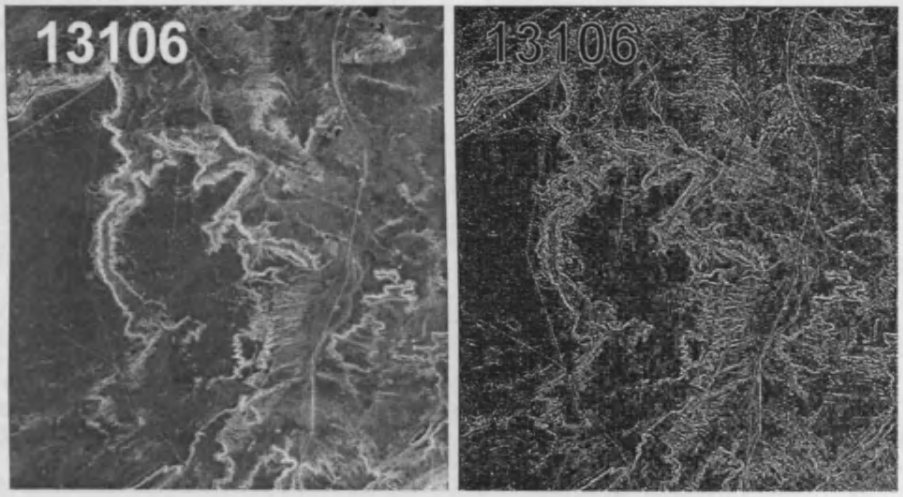


Figure 8.12. Representation of the "13106" test-data image

(left side: original image, right side: after being processed by *Laplacian* Edge detect fixed filter)

```
<?xml version="1.0" ?>
<SARAMETADATA>
  <SARATRACK IDTRACK="13106">
    <NAME>Phnum Voeene, Cambodia</NAME>
    <TRACKDATE>1994-04-16
       00:00:00.0</TRACKDATE>
    <WIDTH>4304</WIDTH>
    <HEIGHT>7996</HEIGHT>
    <CHANNELS>2</CHANNELS>
    <SARACOORDS>
       <SARACOORD>
         <LAT>16.317</LAT>
         <LON>107.654</LON>
       </SARACOORD>
       <SARACOORD>
         <LAT>16.061</LAT>
         <LON>108.082</LON>
       </SARACOORD>
       <SARACOORD>
         <LAT>16.828</LAT>
         <LON>108.575</LON>
       </SARACOORD>
       <SARACOORD>
         <LAT>17.087</LAT>
         <LON>108.144</LON>
       </SARACOORD>
    </SARACOORDS>
    <SARAFILES>
       <SARAFILE NAME="pr13106_byt_hh">
         <POLARIZATION>LHH</POLARIZATION>
       </SARAFILE>
       <SARAFILE NAME="pr13107_byt_hv">
         <POLARIZATION>CHV</POLARIZATION>
       </SARAFILE>
    </SARAFILES>
   <SARASTORED>
      <SERVER>server1</SERVER>
   </SARASTORED>
  </SARATRACK>
</SARAMETADATA>
```

(a)

```
<?xml version="1.0" ?>
<SARAMETADATA>
  <SARATRACK IDTRACK="13106">
    <NAME>Phnum Voeene, Cambodia</NAME>
    <TRACKDATE>1994-04-16
       00:00:00.0</TRACKDATE>
    <WIDTH>4304</WIDTH>
    <HEIGHT>7996</HEIGHT>
    <CHANNELS>2</CHANNELS>
→   <FILTERS>1</FILTERS>
    <SARACOORDS>
       <SARACOORD>
         <LAT>16.317</LAT>
         <LON>107.654</LON>
       </SARACOORD>
       <SARACOORD>
         <LAT>16.061</LAT>
         <LON>108.082</LON>
       </SARACOORD>
       <SARACOORD>
         <LAT>16.828</LAT>
         <LON>108.575</LON>
       </SARACOORD>
       <SARACOORD>
         <LAT>17.087</LAT>
         <LON>108.144</LON>
       </SARACOORD>
    </SARACOORDS>
    <SARAFILES>
       <SARAFILE NAME="pr13106_byt_hh">
         <POLARIZATION>LHH</POLARIZATION>
       </SARAFILE>
       <SARAFILE NAME="pr13107_byt_hv">
         <POLARIZATION>CHV</POLARIZATION>
       </SARAFILE>
    </SARAFILES>
→   <USER_FILES>
       <USER_FILE NAME="pr13106_filter1">
       <FILTER>Laplacian</FILTER>
       <FILTERED_DATE>2004-06-11
          11:02:00.0</FILTERED_DATE>
       <URL_BASE>http://www.cs.cf.ac.uk/
             user/C.Georgousopoulos1/gallium
             /EXSA/010403/</URL_BASE>
       </USER_FILE>
    </USER_FILES>
   <SARASTORED>
      <SERVER>server1</SERVER>
   </SARASTORED>
  </SARATRACK>
</SARAMETADATA>
```

(b)

Code 8.4. Data results

8.4. Conclusion

This chapter has demonstrated how the information and services provided by the SARA active Digital Library may be accessed by a user with a web-based interface, as well as from an external FIPA-complaint agent. Experiments conducted on the SARA prototype developed, show the interactions of

stationary agents with mobile agents for the accomplishment of a request placed by a user/agent from its initial stage to its completion. Distribution of agent load among the information-servers achieved by the management agents is also demonstrated using our algorithm. The adaptability of the LB model has been examined based on three different load balancing schemes, whereas the interoperability of the system has been tested against two different types of agent platforms (FIPA-OS and JADE).

Chapter 9. Future work

9.1. Introduction

This thesis proposed an agent-based architecture for the realisation of an active Digital Library (DL) with emphasis on its interoperability, and with support for load balance of mobile agents within the MAS utilising the DL. A prototype has been developed with reference to the SARA active DL. This chapter suggests further work that remains to be done and which may provide the motivation for new research studies. Discussion on future work is separated into three sections according to the main areas on which this thesis focuses.

9.2. Future work on the SARA agent-based system

Enhancements to the SARA agent-based system may be directly correlated to the actual architecture of the MAS utilising the SARA active DL, or to the interface provided for a user to access the SARA system.

When highly demanding or time-consuming tasks are involved in a system, failure of system components may result in significant loss of information and processing time. This entails the re-execution of the whole task or a part of it. Different fault-tolerance mechanisms exist to provide recovery in case of a server failure, failure of an agent platform environment, the agent itself, network or agent communication break-downs. Usually this is achieved with the use of cloned agents[42][135], monitoring agents[76][92][116][142], replicated tasks performed on multiple hosts concurrently[103][100], check-pointing techniques for successful stage logging[122] etc. A fault tolerance mechanism transparent to the user may be introduced as an extension to the current SARA architecture with the ability to be enabled on-demand from a user, according to its privileges or by the system itself on highly intensive tasks. Note that the current architecture of SARA provides fault-tolerance capabilities for mobile agent migrations, due to URA's Voyager callbacks, and reconstruction of the management agents' SPACE along with automatic recovery of previously failed management agents in SPACE.

Security is another major aspect that has not been addressed in the SARA architecture. Although the realisation of the LSA (Local Security Agent) would restrict users on the access level to information and compute resources (according to their privileges), further security is needed to ensure:

i) a secure web-based interface for the communication between the users and the SARA system, probably using SSL (Secure Sockets Layer) protocol[141].
ii) secure agent communication between external agents and the FIPA-compliant gateway agent(s), and vice-versa; probably achieved by using public key encryption techniques for encrypting/decrypting messages. Further security will be required in the development of a mobility layer for authenticating agent migrations, probably by using authenticated identities and/or object signing.
iii) coherent execution of any custom analysis algorithm provided by a user for further data fusion (without malicious intent, like a virus or a Trojan), carried out on an *isolated* environment with predefined time of utilising a compute server's resources.

The realisation of the LIGA (Local InteGration Agent) which has not been developed in the SARA prototype would enable the breaking up of a query into smaller sub-queries, assigning them to different mobile agents and upon receipt of results, combining and integrating these results to form the complete answer to the original query. In addition, LIGA should provide a gateway to a local cluster or a parallel machine, and ensure that suitable libraries are available on the required server to guarantee execution of a program/code carried by a visiting URA.

Moreover, a monitoring mechanism introduced within the SARA system would be essential for observing and improving the performance and reliability of the system. A monitoring mechanism concerns the collection, analysis and visualisation of information derived from the agents' behaviour within the system, and the servers' utilisation for performance optimisation as well as basic debugging. Instead of incorporating an existing monitoring tool into the SARA system, the management agents' information on the system status and the mobile agents' progress maintained by UMAs may be reused to provide the basic input to a monitoring tool developed for SARA. As in FLASH[71] the monitoring information is directly obtained by using existing agent system features. Moreover, due to the UMA

management agents' information on URAs, caching techniques are possible to be applied. For instance, a counter on an agents recorded queries may be used to identify the most frequent ones (to support caching).

The web-based interface acts as to the front-end to users, and that needs to access the SARA active DL. Further work may include:

i) the design of an interface to enable the collection of SAR images based on a set of coordinates which may be entered manually by the user, or resolved dynamically by the vertices of a polygon surrounding a specific region drawn by the user on the map of Earth.
ii) support for more scientific tasks such as the analysis of mutli-temporal images corresponding to changes in the ecology of a particular region, comparison of SAR images based on phase and amplitude differences of backscatter radiation to study geological events (i.e. motions of ice-sheets or glaciers, seismic or volcano processes), monitoring of a given region in case of natural disasters such as forest fires or flash floods etc.
iii) exploitation of the current agent-based architecture that supports on-demand processing. Apart from the ability of a user to process data against a fixed or custom filter, the web-based interface should provide appropriate operational control for utilising compute server facilities directly from the client-side i.e. the web-based interface. In this instance, a user would be able to steer the processing of data on-the-fly.
iv) different options on data visualisation, achieved by using various types of XSL documents (since results are encoded in XML format), where advanced visualisation will require the employment of specialised visualisation tools, as discussed in *Chapter 2 - section 2.2.2.*
v) monitoring of URA's progress and location.

Finally, the introduction of a User Profile Agent (UPA) would assist the management of a user's profile which may include predefined visualisation settings, maintenance of previous recorded queries and user file-space etc.

9.3. Future work on the interoperability part of SARA architecture

The advantages of agent technology and specially mobile agents has been identified throughout this thesis. The proposed approach to conforming a legacy MAS to a FIPA-compliant one enables agents from different systems to interoperate. The development of a mobility layer as an extension to the FIPA-compliant gateway agent(s) approach supports mobile agent migrations between heterogeneous MASs on various agent platforms.

A mechanism to provide control of agents migrating to different types of agent platforms, in contrast with those migrating to a host of the same agent platform, differs in its complexity. In both cases, a security layer is vital for authenticating agent movements. In this instance, a security layer would also ensure secure agent communication by enabling agents to exchange messages using different encryption protocols.

As in [26] a server houses different kinds of agent platforms to enable agents from architecturally different agent systems to interoperate between themselves, a similar approach may be followed for the realisation of a mobility layer that would enable agent migrations to different types of agent platforms. Alternatively, the development of a mobility layer may be based on the actual architecture of the migrating agent[18]. Another approach is to separate the platform-independent part of an agent from the platform-specific part[97][104], as discussed in *Chapter 3*.

Note that FIPA efforts on "Agent Management Support for Mobility Specification" have been deprecated by FIPA, whereas MASIF restricts the interoperability of agents to those developed on CORBA platforms, see *Chapter 3 - section 3.2.4.1*.

9.4. Future work on the load balance technique in SARA MAS

The load balance mechanism in SARA is a combination of the state-based and model-based approaches. The state-based part of LB is responsible for gathering the system state information, whereas the model-based part of LB controls the distribution of mobile agents within the system based on the information provided and predictions on server utilisations and agent task lifetimes. Therefore, the state-based part of LB deals with the quality, minimisation and timing of collecting the system state information along with

its quick distribution between the management agents, the model has to exploit this information and provide a decision. This is often a complicated process. Therefore, future LB work includes the following:

i) The point at which a server overloads i.e. should not accept further agents, has to be determined. This may be achieved by setting a boundary on a server's CPU utilisation, beyond which the introduction of new agents should be prohibited. In addition, the level of agent-persistence versus slow agent task execution has to be examined. For instance, if a server is close to its overloading limit, would it be worthwhile for an agent task to be executed slower on this server (due to the high utilisation)? What would be the affect on the performance of the other agents executing on the same server, and the overall system?

ii) In the case where an agent task is similar but not exactly the same to a task performed by another agent in the past (referring to 'case 1' of the LB model in *Chapter 6 - section 6.3.2.6*), an algorithm should be developed to decide - according to the level of similarity - if the new agent should extract directly the required information for a part of its task which is exactly the same to the task already or execute its task normally irrespectively. Apart from which method results in the fastest accomplishment of a task, the affect on the system of re-executing part of an already accomplished task instead of working within (filtering) the existing results of a prior task also has to be considered.

iii) The model has to be extended to take into account features other than the processing power like the available memory of a server and the percentage being used, the type of the storage medium employed (e.g. hard/optical disks, tapes) etc. Of course, as with the CPU there should be limit on the number of incoming agents, analogous restrictions should be applied for insufficient memory and simultaneous tape usage.

iv) The estimation of a complex agent task lifetime requires two assumptions regarding the filtering of images (see, *Chapter 6 - section 6.3.2.7*). The first one concerns the processing throughput of each analysis algorithm, and the second one the properties of the image(s) being filtered; which are assumed to be the same in each case. These assumptions may be eliminated and therefore optimize the estimation of a complex agent task by introducing a factor ϕ to the formula that calculates the lifetime of such a task i.e. the updated formula becomes $\phi \cdot \left(S_{data} / P_{comp}\right)$. Here, ϕ is a function of

the properties of the images to be filtered according to their resolution, size, type and the processing performance of the particular analysis algorithm used for filter.

Finally, the progress of the mobile agents within the system, along with the overall system state information should provide a means of improving the intelligence of the management agents for balancing decisions. Possible faults or miscalculations on the proposed model could be identified and modified only by observing and analysing such data.

9.5. Conclusion

This chapter has identified the key elements of the proposed agent-based architecture for the utilisation of the SARA active digital library that need to be extended or amended as part of the future work and research of this thesis. Suggestions on the basic architecture of SARA about securing the information exchange between users and the MAS utilising the DL, fault tolerance/caching/monitoring mechanisms that may be applied, the breaking up of an agent query into smaller sub-queries and the integration of the sub-queries' results to form the complete answer to the original query, as well as improvements on the web-based GUI interface have been reported. Propositions on extending the interoperability of the architecture by defining a security and a mobility layer, to optimisations on the model of the load balancing scheme employed have also been discussed. Although, a few extensions concern just programming-related aspects, most of them may provide the basis for new research studies.

Chapter 10. Conclusion

Remote-sensing data about the Earth's environment is being created at an ever-increasing rate and distributed among heterogeneous remote sites. Traditional model of distributed computing are inadequate to support such complex applications, which generally involve a large quantity of data. The problem of managing such large digital data archives is particularly challenging when the system must cope with data which is processed on demand.

This dissertation proposed a scalable agent-based architecture for the realisation of an active digital library composed of remote-sensing archives, which apart from data-retrieval services provides support for computing services. System Integration and Data Management is achieved based on a set of collaborative agents, where each agent undertakes a pre-defined role responsible for offering a particular type of service. The most complex functionality is localised in non-mobile agents, which remain at one location, providing resources and facilities to lightweight mobile agents that require less processor time to be serialised, and are therefore quicker to transmit. User queries are encapsulated into mobile agents that migrate through the resource servers and interact with the local stationary agents to serve user requests. The utilisation of mobile agents supports autonomous and dynamic on-demand data processing, as well as the transmitting of user developed analysis algorithms to data sources for local fusion. The modularity of the architecture enables existing or new information sources and services to be updated or integrated into the system dynamically. In this instance, if the local archive system of a resource server changes, only the stationary agent that manages the data source that contains the data will need to be amended.

The system architecture does not have a global administrator agent and therefore there is no central point of failure. A management agent exists in every resource server to monitor the local system status and balance the load of mobile agent tasks among the available resource servers by defining their itinerary. In the event of a failure in one of the management agents, the system can operate with all the remaining ones. Management agents exchange between themselves information on their local system status, so that every management agent has a global perspective of the system. The task assignment policy followed by

the management agent is based on a dynamic load balancing scheme derived by the combination of the state-based and model-based approaches.

The objective was to design a load balancing scheme suitable for active archival systems such as digital libraries, by combining the most attractive features of existing load balancing approaches. The outcome was the design of an architecture based on special agents positioned in every server i.e. the management agents, and the derivation of a model that accepts as input parameters an agent's requirements and gives as output the appropriate server(s) where the particular agent should migrate to in order to fulfill its task. This model is incorporated within every management agent that has control over the load balancing decisions. Calculations on the model are based on the system state information maintained by the management agents, estimations on the lifetime of agent tasks and predictions on utilisation of servers. An algorithm has also been developed as an extension to the proposed model, to overcome situations where predictions on lifetime of tasks cannot be estimated or tend to be erroneous. The interactions between the management agents as well as the reuse of this information to support monitoring and caching techniques has also been discussed.

The architecture was further optimised by defining a gateway to provide interoperability with other heterogeneous agent-based systems. Interoperability in the sense that information and services provided by an agent-based system may be utilised by an external system composed of agents operating on a different type of platform. In this instance, information retrieved from the SARA DL may be further enhanced by additional information gathered from a Geographic Information System that is capable of interoperating with SARA. The longitude and latitude of a particular area of the Earth may be used as parameters on an external GIS to retrieve land information such as street names, which can then be combined with SARA image(s) of the corresponding geographical coordinates, resulting in a detailed map of the particular area. Although we are aware that coordinate systems between GIS systems do not always overlap, we assume that all the systems that use our agents make use of a similar set of metadata to specify coordinates. The coordinate system we utilize is based on a standard adopted by NASA.

The architecture of the gateway approach is generic and may be easily adopted by any system operating data archival services. Using our system, a developer does not have to have any knowledge of the FIPA

standards specifications for conforming a legacy MAS to a FIPA-compliant one. Consequently time may be saved in terms of reading, understanding, and applying the FIPA specifications to a MAS that needs to be FIPA compliant. Although the default gateway is limited in scope, as not all FIPA performatives are supported, if more complex interaction is necessary, such as negotiation, co-operation or co-ordination of heterogeneous agents, it is possible to extend the gateway to achieve this. This will require limited knowledge of FIPA specifications regarding the ACL message structure and the performative(s) that need to be defined using the template provided in the GatewayAgent API we specify.

The development of a prototype with reference to the SARA digital library has been used as a test-bed for experimental tests to assess and validate the reliability of the proposed architecture, and the principles, ideas and propositions expressed within this dissertation. The structure of every entity involved in the system has been discussed and the most important implementation considerations of the prototype have been outlined.

Experiments have shown the successful accomplishment of different user queries performed by mobile agents in collaboration with the stationary agents of the visited resource servers. A simple query involved the acquisition of data composed of a collection of SAR images defined by specific coordinates. More complex queries involved the filtering of data retrieved from the execution of simple query, against an image analysis algorithm maintained on a compute server of the digital library, or a custom one that had to be transferred by a mobile agent (to the server). The interoperability of the architecture has been tested with two different types of agent platforms (FIPA-OS and JADE) on the successful accomplishment of requests related with information retrieval from the digital library. The simultaneous launch of multiple agents demonstrates how the management agents can support the even distribution of agent load among the available servers (that constituted the digital library). Experiments to explore the efficiency of the proposed LB model, based on the ability to amend the utilisation of a server when it is miscalculated due to error-estimations on agent task lifetimes, provided an optimisation of up to 10.8% in load balancing. In conclusion, the key elements of the proposed agent-based architecture for the utilisation of an active digital library that may be extended as part of future work of this research, which may provide the motivation for new research studies, have also been discussed.

Appendix

A1. Database test-data

The test-data set that has been used in the conduction of experiments in the SARA prototype is composed of 60 elements. The information of those elements is stored in a relational database consisted of four tables, maintained by the Oracle DBMS. Figure 4.3 of *Chapter 4 - section 4.4* illustrates the EAR of the database revealing the entities of each table and the relationships between themselves. The test-data set is a small representative set of data acquired by the SIR-C shuttle mission in 1994/95 and have been obtained from the SARA server of the University of Leece in Italy[129].

The four tables provided below maintain information for each SAR image of the test-data set. The *Track* table houses information for each image such as its name, date of acquisition, unique id, width, height and number of channels. The *Coords* table contains the latitude and longitude coordinates of the four vertex of image. In the *File* table the filenames of all SAR images along with their versions of different polarization are recorded, and finally the *Stored* table contains information of where each image is actually stored.

Table A1. STORED table

SERVER	IDTRACK	SERVER	IDTRACK	SERVER	IDTRACK	SERVER	IDTRACK
server1	44829	server1	42844	server1	43040	server2	41117
server1	03432	server2	42844	server2	43040	server2	41514
server1	11577	server1	42846	server1	44819	server2	41866
server1	11839	server2	42846	server2	44819	server2	41986
server1	11841	server1	42848	server1	44827	server2	42056
server1	11842	server2	42848	server2	44827	server2	42228
server1	11990	server1	42850	server1	14112	server2	42525
server1	12401	server2	42850	server2	14112	server2	42527
server1	13100	server1	42864	server1	11478	server2	42593
server1	13106	server2	42864	server2	11478	server2	42627
server1	13110	server1	42948	server2	13190	server2	42738
server1	13156	server2	42948	server2	13192	server2	42740
server1	13158	server1	43000	server2	13194	server2	42742
server1	13160	server2	43000	server2	13196	server2	42754
server1	13162	server1	43002	server2	14321	server2	42810
server1	13164	server2	43002	server2	14325	server2	42812
server1	13184	server1	43036	server2	14327	server2	42836
server2	42840	server2	43036	server2	14501	server2	42838

Table A2. COORDS table

IDTRACK	V1_LAT	V1_LONG	V2_LAT	V2_LONG	V3_LAT	V3_LONG	V4_LAT	V4_LONG
42527	11.947	104.579	12.121	104.878	11.343	105.345	11.17	105.048
42525	13.072	103.895	13.247	104.195	12.471	104.667	12.296	104.368
42754	13.466	103.653	13.642	103.954	12.867	104.427	12.691	104.128
13100	13.96	6.165	13.704	106.596	14.475	107.077	14.734	106.643
12401	16.286	107.499	16.103	107.801	16.867	108.298	17.051	107.995
13106	16.317	107.654	16.061	108.082	16.828	108.575	17.087	108.144
13110	17.824	108.632	17.559	109.072	18.323	109.574	18.591	109.131
42742	18.125	100.729	18.305	101.034	17.537	101.531	17.358	101.227
42740	18.904	100.217	19.086	100.524	18.319	101.025	18.139	100.72
42738	19.349	99.922	19.532	100.23	18.766	.734	18.584	100.428
13190	29.825	46.774	30.027	46.448	30.761	47.054	30.558	47.382
13184	30.212	47.076	30.412	46.755	31.143	47.369	30.941	47.691
13192	30.52	47.35	30.723	47.022	31.455	47.638	31.249	47.967
13194	31.271	47.968	31.475	47.64	32.203	48.266	31.996	48.595
13196	31.906	48.517	32.113	48.188	32.838	48.822	32.629	49.153
13162	33.132	-115.196	33.501	-114.607	32.775	-113.969	32.409	-114.555
13164	34.012	-115.104	34.421	-114.461	33.701	-113.805	33.297	-114.446
42848	34.253	-119.307	34.049	-119.019	34.731	-118.313	34.937	-118.602
42840	34.268	-119.816	34.114	-119.602	35.793	-117.891	35.948	-118.105
11839	34.3	261.126	34.131	261.365	35.49	262.792	35.663	262.552
42850	34.848	-118.695	34.636	-118.398	35.313	-117.681	35.528	-117.979
41514	34.87	-118.7	33.51	-117.29	33.2	-117.71	34.56	-119.12
42838	34.96	-119.093	34.805	-118.879	35.477	-118.154	35.634	-118.369
42846	35.022	-118.857	34.344	-118.143	34.036	-118.57	34.712	-119.283
11990	35.104	-98.068	35.007	-98.205	34.665	-97.851	34.761	-97.714
42836	35.549	-118.462	35.392	-118.247	36.06	-117.511	36.218	-117.726
42844	35.561	-119.436	34.887	-118.713	34.577	-119.139	35.248	-119.863
42228	37.214	-117.748	36.993	-117.451	37.646	-116.681	37.87	-116.978
14112	40.794348	15.912602	40.63585	16.122002	41.267825	16.962351	41.428104	16.75326
11478	40.959	15.971	41.279	16.371	41.08	16.633	40.764	16.232
03432	41.500448	16.029775	41.309379	15.755796	40.650606	16.55554	40.839589	16.829289
42627	43.773182	57.674191	44.135632	58.199341	43.478062	59.054256	43.119942	58.529652
42593	43.834	53.87	43.583	53.57	42.99	54.492	43.237	54.792
42948	43.863	51.277	43.479	51.708	44.038	52.673	44.427	52.245
43000	47.95	10.711	48.502	11.887	48.337	12.056	47.787	10.882
43002	47.95	10.711	48.502	11.887	48.337	12.56	47.787	10.882
43036	47.95	10.711	48.502	11.887	48.337	12.056	47.787	10.882
44819	47.968	10.68	48.464	11.906	48.282	12.066	47.789	10.844
43040	48.005	10.698	48.504	11.921	48.282	12.119	47.786	10.9
14325	48.038	10.801	48.565	12.002	48.341	12.217	47.817	11.019
43042	48.042	10.859	48.535	12.089	48.353	12.248	47.862	11.022
14577	48.09	10.964	49.141	13.275	48.944	13.47	47.899	11.164
11577	48.098	10.959	48.328	11.525	48.112	11.715	47.883	11.151
41986	48.106	11.038	48.337	11.611	48.162	11.767	47.931	11.196
14327	48.115	10.587	48.612	11.815	48.173	12.204	47.681	10.983
42810	49.234	-97.622	49.041	-97.441	49.274	-96.858	49.468	-97.037
13156	49.24	-97.688	49.007	-97.47	49.24	-96.887	49.474	-97.103
13158	49.268	-97.629	49.026	-97.41	49.252	-96.822	49.495	-97.038
42812	49.276	-97.698	48.982	-97.431	49.209	-96.844	49.505	-97.108
13160	49.358	-97.767	48.895	-97.386	49.104	-96.786	49.571	-97.162
14501	51.325	12.495	50.866	13.833	50.695	13.683	51.151	12.349
44829	51.628	11.484	51.182	12.842	51.037	12.719	51.481	11.364
44827	51.628	11.484	51.182	12.842	51.037	12.719	51.481	11.364
14321	51.703	11.282	51.259	12.645	51.098	12.509	51.54	11.15
41866	52.229	-1.511	52.565	-.154	52.138	.117	51.806	-1.227
41117	52.41	-.13	52.73	1.22	52.56	1.32	52.24	-.02
11841	54.648	12.565	54.384	12.649	54.535	14.161	54.801	14.088
42056	54.698	13.105	54.468	13.031	54.384	13.785	54.614	13.863
11842	55.129	12.622	54.618	12.462	54.449	13.972	54.958	14.153
42864	55.924	37.314	55.702	37.369	55.816	38.945	56.039	38.899

Table A3. IDTRACK table

IDTRACK	NAME	DATE_AQ	WIDTH	HEIGHT	CHANNELS
44829	Harz, Germany:LEIPZIG, GERMANY	NULL	1448	8555	8
03432	Puglia, Italy	11-APR-94	2500	2500	1
11577	Oberpfaffenhofen, Germany:OBERPFAFFENHOFEN, GERMANY	NULL	3524	9608	4
11839	Chickasha, Oklahoma	16-APR-94	2316	15996	4
11841	Thetford, England	16-APR-94	2388	7996	4
11842	North Sea A2	16-APR-94	4608	7996	2
11990	Chickasha, Oklahoma	16-APR-94	1456	11144	8
12401	Hainan, China	16-APR-94	1524	3997	4
13100	Phnum Voeene, Cambodia	16-APR-94	4360	7995	4
13106	Phnum Voeene, Cambodia	16-APR-94	4304	7996	4
13110	Phnum Voeene, Cambodia	16-APR-94	4404	7995	4
13156	Altona, Manitoba Canada	16-APR-94	1456	11552	8
13158	Altona, Manitoba Canada	16-APR-94	1684	8980	8
13160	Altona, Manitoba Canada	16-APR-94	3716	9616	4
13162	Owens Valley, California	16-APR-94	5456	7995	4
13164	Stovepipe Wells, California	16-APR-94	5980	7995	4
13184	Saudi Arabia C	16-APR-94	3036	7994	4
13190	Saudi Arabia C	16-APR-94	3088	7995	4
13192	Saudi Arabia C	16-APR-94	3096	7995	4
13194	Saudi Arabia C	16-APR-94	3084	7994	4
13196	Saudi Arabia C	16-APR-94	3088	7995	4
14321	Harz, Germany:HALLE SAALE, GERMANY	NULL	1604	8555	8
14325	Oberpfaffenhofen, Germany:	NULL	2348	8555	8
14327	Oberpfaffenhofen, Germany:	NULL	4516	8555	4
14501	Harz, Germany:FREIBERG, GERMANY	NULL	1724	8555	8
14577	Oberpfaffenhofen,Germany:ERDING,GERMANY (DBLSCENE)	NULL	1032	8277	8
41117	Thetford, England	NULL	1628	7996	8
41514	Los Angeles, California, US	NULL	4152	15995	4
41866	Thetford, England	NULL	4064	7995	4
41986	Oberpfaffenhofen, Germany:	NULL	5676	10408	2
42056	North Sea A0	16-APR-94	1388	10428	8
42228	N Grapevine Mtns 1, CA	16-APR-94	2864	7995	8
42525	Angkor Wat, Cambodia	16-APR-94	3036	7996	4
42527	Angkor Wat, Cambodia	16-APR-94	3024	7995	4
42593	Turkmenistan	16-APR-94	2932	7995	4
42627	Almaz 5, Russia	16-APR-94	4672	7996	4
42738	Angkor Wat, Cambodia	16-APR-94	3056	7996	4
42740	Angkor Wat, Cambodia	16-APR-94	3052	7995	4
42742	Angkor Wat, Cambodia	16-APR-94	3040	7995	4
42754	Angkor Wat, Cambodia	16-APR-94	3036	7996	4
42810	Altona, Manitoba Canada	16-APR-94	1212	11552	8
42812	Altona, Manitoba Canada	16-APR-94	2040	8692	8
42836	Stovepipe Wells, California	16-APR-94	2088	7996	8
42838	Stovepipe Wells, California	16-APR-94	2084	7995	8
42840	Stovepipe Wells, California	16-APR-94	2084	7996	8
42844	Los Angeles, California, US:TAFT, CALIFORNIA	NULL	4156	7995	4
42846	Los Angeles, California, US:PIRU LAKE, CALIFORNIA	NULL	4156	7996	4
42848	Stovepipe Wells, California	16-APR-94	2788	7995	4
42850	Stovepipe Wells, California	16-APR-94	2876	7994	4
42864	Medwez region, Russia	16-APR-94	1984	7995	8
42948	Zhamanshin, USSR	16-APR-94	4396	7996	3
43000	Oberpfaffenhofen, Germany:	NULL	1748	8556	8
43002	Oberpfaffenhofen, Germany:	NULL	1748	8556	4
43036	Oberpfaffenhofen, Germany:	NULL	1748	8556	8
43040	Oberpfaffenhofen, Germany:	NULL	2288	8555	4
43042	Oberpfaffenhofen, Germany:	NULL	1868	8555	2
44819	Oberpfaffenhofen, Germany:	NULL	1868	8555	2
44827	Harz, Germany:LEIPZIG, GERMANY	NULL	1448	8555	8
14112	Puglia, Italy	16-APR-94	2000	7900	1
11478	Matera, Italy:MATERA, ITALY	NULL	1212	9780	8

Table A4. FILE table

IDTRACK	NAME	POL
44829	pr44829_byt_hh	LHH
44829	pr44829_byt_hv	LHV
44829	pr44829_byt_vh	LVH
44829	pr44829_byt_vv	LVV
44829	pr44830_byt_hh	CHH
44829	pr44830_byt_hv	CHV
44829	pr44830_byt_vh	CVH
44829	pr44830_byt_vv	CVV
11577	pr11577_byt_vh	LVH
11577	pr11577_byt_vv	LVV
11577	pr11578_byt_vh	CVH
11577	pr11578_byt_vv	CVV
11839	pr11839_byt_hh	LHH
11839	pr11839_byt_hv	LHV
11839	pr11839_byt_vh	LVH
11839	pr11839_byt_vv	LVV
11841	pr11841_byt_hh	LHH
11841	pr11841_byt_hv	LHV
11841	pr11841_byt_vh	LVH
11841	pr11841_byt_vv	LVV
11842	pr11842_byt_vv	LVV
11842	pr11843_byt_vv	CVV
11990	pr11990_byt_hh	LHH
11990	pr11990_byt_hv	LHV
11990	pr11990_byt_vh	LVH
11990	pr11990_byt_vv	LVV
11990	pr11991_byt_hh	CHH
11990	pr11991_byt_hv	CHV
11990	pr11991_byt_vh	CVH
11990	pr11991_byt_vv	CVV
12401	pr12401_byt_hh	LHH
12401	pr12401_byt_hv	LHV
12401	pr12402_byt_hh	CHH
13100	pr13100_byt_hh	LHH
13100	pr13100_byt_hv	LHV
13100	pr13101_byt_hh	CHH
13100	pr13101_byt_hv	CHV
13106	pr13106_byt_hh	LHH
13106	pr13106_byt_hv	LHV
13106	pr13107_byt_hh	CHH
13106	pr13107_byt_hv	CHV
13110	pr13110_byt_hh	LHH
13110	pr13110_byt_hv	LHV
13110	pr13111_byt_hh	CHH
13110	pr13111_byt_hv	CHV
13156	pr13156_byt_hh	LHH
13156	pr13156_byt_hv	LHV
13156	pr13156_byt_vh	LVH
13156	pr13156_byt_vv	LVV
13156	pr13157_byt_hh	CHH
13156	pr13157_byt_hv	CHV
13156	pr13157_byt_vh	CVH
13156	pr13157_byt_vv	CVV
13158	pr13158_byt_hh	LHH
13158	pr13158_byt_hv	LHV
13158	pr13158_byt_vh	LVH
13158	pr13158_byt_vv	LVV
13158	pr13159_byt_hh	CHH
13158	pr13159_byt_hv	CHV
13158	pr13159_byt_vh	CVH
13158	pr13159_byt_vv	CVV

IDTRACK	NAME	POL
14325	pr14325_byt_vh	LVH
14325	pr14325_byt_vv	LVV
14325	pr14326_byt_hh	CHH
14325	pr14326_byt_hv	CHV
14325	pr14326_byt_vh	CVH
14325	pr14326_byt_vv	CVV
14327	pr14327_byt_hh	LHH
14327	pr14327_byt_hv	LHV
14327	pr14328_byt_hh	CHH
14327	pr14328_byt_hv	CHV
14501	pr14501_byt_hh	LHH
14501	pr14501_byt_hv	LHV
14501	pr14501_byt_vh	LVH
14501	pr14501_byt_vv	LVV
14501	pr14502_byt_hh	CHH
14501	pr14502_byt_hv	CHV
14501	pr14502_byt_vh	CVH
14501	pr14502_byt_vv	CVV
14577	pr14577_byt_vv	LVV
14577	pr14578_byt_hh	CHH
14577	pr14578_byt_hv	CHV
14577	pr14578_byt_vh	CVH
14577	pr14578_byt_vv	CVV
14577	pr14577_byt_vh	LVH
14577	pr14577_byt_hh	LHH
14577	pr14577_byt_hv	LHV
41117	pr41117_byt_hh	LHH
41117	pr41117_byt_hv	LHV
41117	pr41117_byt_vh	LVH
41117	pr41117_byt_vv	LVV
41117	pr41118_byt_hh	CHH
41117	pr41118_byt_hv	CHV
41117	pr41118_byt_vh	CVH
41117	pr41118_byt_vv	CVV
41514	pr41514_byt_hh	LHH
41514	pr41514_byt_hv	LHV
41514	pr41515_byt_hh	CHH
41514	pr41515_byt_hv	CHV
41866	pr41866_byt_hh	LHH
41866	pr41866_byt_hv	LHV
41866	pr41867_byt_hh	CHH
41866	pr41867_byt_hv	CHV
41986	pr41986_byt_vv	LVV
41986	pr41987_byt_vv	CVV
42056	pr42056_byt_hh	LHH
42056	pr42056_byt_hv	LHV
42056	pr42056_byt_vh	LVH
42056	pr42056_byt_vv	LVV
42056	pr42057_byt_hh	CHH
42056	pr42057_byt_hv	CHV
42056	pr42057_byt_vh	CVH
42056	pr42057_byt_vv	CVV
42228	pr42228_byt_hh	LHH
42228	pr42228_byt_hv	LHV
42228	pr42228_byt_vh	LVH
42228	pr42228_byt_vv	LVV
42228	pr42229_byt_hh	CHH
42228	pr42229_byt_hv	CHV
42228	pr42229_byt_vh	CVH
42228	pr42229_byt_vv	CVV
42525	pr42525_byt_hh	LHH

IDTRACK	NAME	POL
42812	pr42813_byt_hh	CHH
42812	pr42813_byt_hv	CHV
42812	pr42813_byt_vh	CVH
42812	pr42813_byt_vv	CVV
42836	pr42836_byt_hh	LHH
42836	pr42836_byt_hv	LHV
42836	pr42836_byt_vh	LVH
42836	pr42836_byt_vv	LVV
42836	pr42837_byt_hh	CHH
42836	pr42837_byt_hv	CHV
42836	pr42837_byt_vh	CVH
42836	pr42837_byt_vv	CVV
42838	pr42838_byt_hh	LHH
42838	pr42838_byt_hv	LHV
42838	pr42838_byt_vh	LVH
42838	pr42838_byt_vv	LVV
42838	pr42839_byt_hh	CHH
42838	pr42839_byt_hv	CHV
42838	pr42839_byt_vh	CVH
42838	pr42839_byt_vv	CVV
42840	pr42840_byt_hv	LHV
42840	pr42840_byt_vh	LVH
42840	pr42840_byt_vv	LVV
42840	pr42841_byt_hh	CHH
42840	pr42841_byt_hv	CHV
42840	pr42841_byt_vh	CVH
42840	pr42841_byt_vv	CVV
42844	pr42844_byt_hh	LHH
42844	pr42844_byt_hv	LHV
42844	pr42845_byt_hh	CHH
42844	pr42845_byt_hv	CHV
42846	pr42846_byt_hh	LHH
42846	pr42846_byt_hv	LHV
42846	pr42847_byt_hh	CHH
42846	pr42847_byt_hv	CHV
42848	pr42848_byt_hh	LHH
42848	pr42848_byt_hv	LHV
42848	pr42849_byt_hh	CHH
42848	pr42849_byt_hv	CHV
42850	pr42850_byt_hh	LHH
42850	pr42850_byt_hv	LHV
42850	pr42851_byt_hh	CHH
42850	pr42851_byt_hv	CHV
42864	pr42865_byt_vv	CVV
42864	pr42864_byt_hh	LHH
42864	pr42864_byt_hv	LHV
42864	pr42864_byt_vh	LVH
42864	pr42864_byt_vv	LVV
42864	pr42865_byt_hh	CHH
42864	pr42865_byt_hv	CHV
42864	pr42865_byt_vh	CVH
42948	pr42948_byt_hh	LHH
42948	pr42948_byt_hv	LHV
42948	pr42949_byt_hv	CHV
43000	pr43000_byt_hh	LHH
43000	pr43000_byt_hv	LHV
43000	pr43000_byt_vh	LVH
43000	pr43000_byt_vv	LVV
43000	pr43001_byt_hh	CHH
43000	pr43001_byt_hv	CHV
43000	pr43001_byt_vh	CVH

13160	pr13160_byt_hh	LHH
13160	pr13160_byt_hv	LHV
13160	pr13161_byt_hh	CHH
13160	pr13161_byt_hv	CHV
13162	pr13162_byt_hh	LHH
13162	pr13162_byt_hv	LHV
13162	pr13163_byt_hh	CHH
13162	pr13163_byt_hv	CHV
13164	pr13164_byt_hh	LHH
13164	pr13164_byt_hv	LHV
13164	pr13165_byt_hh	CHH
13164	pr13165_byt_hv	CHV
13184	pr13184_byt_hh	LHH
13184	pr13184_byt_hv	LHV
13184	pr13185_byt_hh	CHH
13184	pr13185_byt_hv	CHV
13190	pr13190_byt_hh	LHH
13190	pr13190_byt_hv	LHV
13190	pr13191_byt_hh	CHH
13190	pr13191_byt_hv	CHV
13192	pr13192_byt_hh	LHH
13192	pr13192_byt_hv	LHV
13192	pr13193_byt_hh	CHH
13192	pr13193_byt_hv	CHV
13194	pr13194_byt_hh	LHH
13194	pr13194_byt_hv	LHV
13194	pr13195_byt_hh	CHH
13194	pr13195_byt_hv	CHV
13196	pr13196_byt_hh	LHH
13196	pr13196_byt_hv	LHV
13196	pr13197_byt_hh	CHH
13196	pr13197_byt_hv	CHV
14321	pr14321_byt_vv	LVV
14321	pr14322_byt_hh	CHH
14321	pr14322_byt_hv	CHV
14321	pr14322_byt_vh	CVH
14321	pr14322_byt_vv	CVV
14321	pr14321_byt_vh	LVH
14321	pr14321_byt_hh	LHH
14321	pr14321_byt_hv	LHV
14325	pr14325_byt_hh	LHH
14325	pr14325_byt_hv	LHV

42525	pr42525_byt_hv	LHV
42525	pr42526_byt_hh	CHH
42525	pr42526_byt_hv	CHV
42527	pr42527_byt_hh	LHH
42527	pr42528_byt_hh	CHH
42527	pr42528_byt_hv	CHV
42593	pr42593_byt_hh	LHH
42593	pr42593_byt_hv	LHV
42593	pr42594_byt_hh	CHH
42593	pr42594_byt_hv	CHV
42627	pr42627_byt_hh	LHH
42627	pr42627_byt_hv	LHV
42627	pr42628_byt_hh	CHH
42627	pr42628_byt_hv	CHV
42738	pr42738_byt_hh	LHH
42738	pr42738_byt_hv	LHV
42738	pr42739_byt_hh	CHH
42738	pr42739_byt_hv	CHV
42740	pr42740_byt_hh	CHH
42740	pr42740_byt_hv	CHV
42740	pr42741_byt_hh	LHH
42740	pr42741_byt_hv	LHV
42742	pr42743_byt_hv	CHV
42742	pr42742_byt_hh	LHH
42742	pr42742_byt_hv	LHV
42742	pr42743_byt_hh	CHH
42754	pr42754_byt_hh	LHH
42754	pr42754_byt_hv	LHV
42754	pr42755_byt_hh	CHH
42754	pr42755_byt_hv	CHV
42810	pr42810_byt_hh	LHH
42810	pr42810_byt_hv	LHV
42810	pr42810_byt_vh	LVH
42810	pr42810_byt_vv	LVV
42810	pr42811_byt_hh	CHH
42810	pr42811_byt_hv	CHV
42810	pr42811_byt_vh	CVH
42810	pr42811_byt_vv	CVV
42812	pr42812_byt_hh	LHH
42812	pr42812_byt_hv	LHV
42812	pr42812_byt_vh	LVH
42812	pr42812_byt_vv	LVV

43000	pr43001_byt_vv	CVV
43002	pr43002_byt_hh	LHH
43002	pr43002_byt_hv	LHV
43002	pr43003_byt_hh	CHH
43002	pr43003_byt_hv	CHV
43036	pr43036_byt_hh	LHH
43036	pr43036_byt_hv	LHV
43036	pr43036_byt_vh	LVH
43036	pr43036_byt_vv	LVV
43036	pr43037_byt_hh	CHH
43036	pr43037_byt_hv	CHV
43036	pr43037_byt_vh	CVH
43036	pr43037_byt_vv	CVV
43040	pr43040_byt_hh	LHH
43040	pr43040_byt_hv	LHV
43040	pr43041_byt_hh	CHH
43040	pr43041_byt_hv	CHV
43042	pr43043_byt_vv	CVV
43042	pr43042_byt_vv	LVV
44819	pr44819_byt_vv	LVV
44819	pr44820_byt_vv	CVV
44827	pr44827_byt_hh	LHH
44827	pr44827_byt_hv	LHV
44827	pr44827_byt_vh	LVH
44827	pr44827_byt_vv	LVV
44827	pr44828_byt_hh	CHH
44827	pr44828_byt_hv	CHV
44827	pr44828_byt_vh	CVH
44827	pr44828_byt_vv	CVV
11478	pr11478_byt_hh	LHH
11478	pr11478_byt_hv	LHV
11478	pr11478_byt_vh	LVH
11478	pr11478_byt_vv	LHV
11478	pr11479_byt_hh	CHH
11478	pr11479_byt_hv	CHV
11478	pr11479_byt_vh	CVH
11478	pr11479_byt_vv	CVV
03432	pr03432_byt	Xband
12401	pr12402_byt_hv	CHV
42527	pr42527_byt_hv	LHV
14112	pr14112_byt	Xband
42840	pr42840_byt_hh	LHH

A2. Gateway setup script

The *gateway_*setup script installs and configures the FIPA-OS toolkit on behalf of a developer. The configuration of the FIPA-OS toolkit involves:

1) platform configuration which contains:

i. *platform profile*:

Describes information about the FIPA-OS platform, including the platform's host-name, the *location* of the AMS and location of other profiles used by entities within the platform (i.e. agents, ACC). The identification of a Naming Service (NS) is also necessary for agents on a platform to locate one another. The available FIPA-OS transports supported internally by the agents are the *ssl-rmi* (RMI over SSL) and *fipaos-rmi*. The fipaos-rmi transports can only be used by homogeneous FIPA-OS platforms.

ii. *ACC profile*:

The ACC is the gateway to remote platforms and it is only required if the platform has to interact with other agent platforms. Although, if an agent platform is distributed to more than one machines then the ACC profile should provide the internal transport (internal MTP) that the platform is using.

iii. *default agent profile*:

It provides default agent configuration information for agents without a profile, such as a database type for recording agent conversations, threadpool management features (that concerns the control of the thread pools utilised by the TaskManager in order to execute tasks), and protocol mappings (i.e. specify the mapping between message protocols and the concrete implementations of protocols that the platform should use).

iv. *AgentLoader profile*:

It enables the editing of the list of agents that will be available in the AgentLoader, the agent name (class) which will be instantiated and whether is started automatically when the platform is booted.

v. *installation properties*:

Concerns installation location, profiles' location, version of the toolkit, whether the AgentLoader is started with a GUI, screen and file debug levels.

vi. *aggressive garbage collection*:

Concerns advanced garbage collection features such as garbage collection (GC) delays, memory limits, Java VM settings related with GC.

2) inter-platform configuration contains:

i. *ACC profile*:

Provides configuration information for the ACC of a platform, including which MTPs the platform is using, additional external MTPs it should use for inter-platform communication, details of other platforms that should be contacted at start-up and details of which database type to use to store this information.

ii. *Naming Services*:

The transports used by the ACC for external communication to other platforms are the *corbaname* or *http* transport protocol.

The FIPA-OS configuration files i.e. *profiles* are stored in XML format which are possible to be altered manually after the initial installation. For instance, the ACC's configuration details are maintained in the *acc.profile* file. The following two example XML documents show the *platform* and *ACC* profile of the SARA FIPA-compliant gateway that has been setup on one of the SARA web servers, with the host-name *gallium.cs.cf.ac.uk*.

```
<?xml version="1.0" encoding="UTF-8"?>
<?enhydra-unmarshall package="fipaos.agent.profile"?>
<platformProfile profileDirectory="\fipaos\profiles\"
                 hAPName="gallium.cs.cf.ac.uk"
                 aMSAddress="fipaos-rmi://gallium.cs.cf.ac.uk:3000/ams"
                 dynamic="false" mobility="false"/>
```

Code A1. Platform profile *('platform.profile' filename)*

Those configuration files identify the platform name, the internal/external MTP used along with the IP addresses and ports for communication, the location of the AMS and DF FIPA-agents and details of the external FIPA-compliant MASs that the SARA system can interoperate with; in this case the MAS operating on bloodstone.cs.cf.ac.uk.

```
<?xml version="1.0" encoding="UTF-8"?>
<?enhydra-unmarshall package="fipaos.agent.profile"?>
<aCCProfile localAddressesLocation="\fipaos\platform.addresses">
  <databaseProfile databaseType="SerializationDatabase"
                   databaseLocation="\fipaos\databases\" />
  <internalAddress address="fipaos-rmi://gallium.cs.cf.ac.uk:3000" />
  <externalAddress address="http://gallium.cs.cf.ac.uk:8080" />
  <externalAddress address="corbaname://gallium.cs.cf.ac.uk:4000" />
  <remoteAgentPlatformProfile hAPName="bloodstone.cs.cf.ac.uk"
                              addressesLocation="http://bloodstone.cs.cf.ac.uk:8080/acc" />
</aCCProfile>
```

Code A2. ACC profile *('acc.profile' filename)*

A3. GatewayAgent API

The GatewayAgent API (Application Program Interface) is composed of two library classes, the *GatewayAgent* and the *GAparse*. The *GatewayAgent* library contains a constructor for creating a gateway agent and a number of methods for its setup and maintenance, whereas the *GAparse* contains a single method for parsing an XML document. Actually, this method could be used by the gateway agent to validate an XML document against an ontology defined by a DTD document. Below there is a detailed specification of the GatewayAgent API.

The methods of the *GatewayAgent* can be grouped into four categories according to their functionality.

Constructor summary
GatewayAgent(String platform, String name, String ownership) Construct a new gateway agent

Method summary	
void	**disable**() Disables the gateway agent and frees-up resources
String	**getID**(GatewayAgent GatewayAgent_name) Returns the unique ID of a gateway agent - specified by the *GatewayAgent_name*
boolean	**isEnabled**(GatewayAgent GatewayAgent_name) Returns *true* if a gateway agent is alive - specified by the *GatewayAgent_name*

void	**addProperty**(LinkedList property_details) Adds a property to the gateway agent - specified by the *property_details*
void	**delProperty**(String service_name) Removes a property from the gateway agent - indicated by the *service_name*
LinkedList	**getProperty**(String service_name) Returns a property of the gateway agent - specified by the *service_name*
LinkedList	**getProperties**() Returns all the properties of the gateway agent (from all the available services)
int	**getPropertiesSize**() Returns the number of the gateway agent's properties
void	**updProperty**(String service_name, LinkedList property_details) Updates a property of the gateway agent - indicated by the *service_name*, the

	property_details specify the property details to be updated

void	**delPerformative**(String performative_name) Removes a performative from the gateway agent's list of supported performatives - indicated by the *performative_name*
int	**getNumberOfPerformatives**() Returns the number of performatives supported by the gateway agent
LinkedList	**getPerformatives**() Returns a list of the performatives supported by the gateway agent
void	**setPerformative**(String performative_name) Adds a performative to the gateway agent's list of supported performatives - specified by the *performative_name*

void	**delEXservice**(String ex_service_name, String ex_DF_name) Removes the external service specified by *ex_service_name* from the configuration details of the gateway agent. The *ex_DF_name* indicates the DF of the MAS hosting the service to be removed
LinkedList	**getEXservices**() Returns a list of the external services stored in the gateway agent's configuration details
void	**setEXservices**(String ex_DF_name, LinkedList service_name, String com_protocol) Sets a list of external services specified by the *service_name* parameter, the DF's name of the MAS which hosts the specified services indicated by the *ex_DF_name* parameter and a list of the communication protocols supported by the corresponding MAS
String	**sendRequest**(String ex_service_name, String message) (gateway agent) sends a REQUEST to the external agent that provides the service indicated by *ex_service_name* with content as the content of the *message* parameter

, and the specification of the *GAParse* library are:

Constructor summary
GAparse() Construct a new parser

Method summary	
boolean	**parseXML_DTD**(String xml_filename, String dtd_ontology_filename) Returns *true* if an XML document is parsed flawless against a DTD document

A4. Image processing filters

The image processing of digital images enhance and reveal hidden information. The implementation of the fixed filters developed for the SARA prototype is based on a process called *convolution*[145]. Convolution is a spatial operation that computes each output sample by multiplying elements of a *kernel* with the samples surrounding a particular source sample. Convolution filtering operates on a group of input pixels surrounding a center one. The adjoining pixels provide important information about brightness trends in the area of the pixel being processed. Convolution filtering moves across the source image (pixel by pixel) vertically or horizontally, placing resulting pixels into the destination image. The resulting brightness of each source pixel depends on the group of pixels surrounding the source pixel. Using the brightness information of the source pixel's neighbors, the convolution process calculates the spatial frequency activity in the area, making it possible to filter the brightness based on the spatial frequency of the area. Convolution filtering uses a *convolve kernel*, containing an array of convolution coefficient values, called *key elements*, as illustrated in Figure A1.

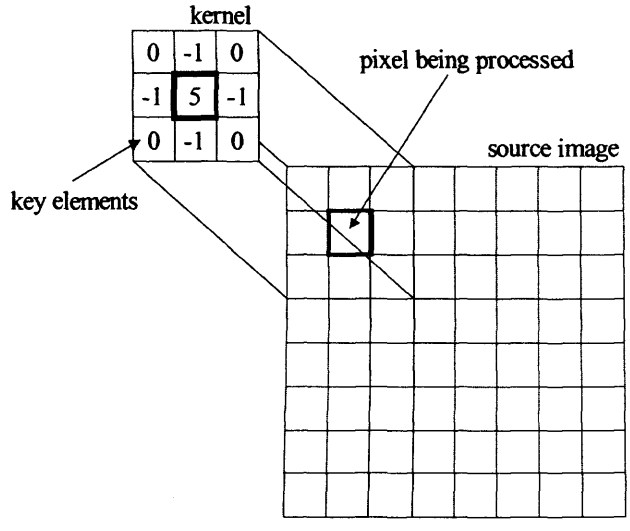


Figure A1. Convolve kernel[145]

The array is not restricted to any particular size, and does not even have to be square. The kernel can be 1x1, 3x3, 5x5, MxN. A larger kernel size affords a more precise filtering operation by increasing the number of neighboring pixels used in the calculation. The larger the kernel is, the more computations that are required to be performed. For instance, an image of 640x480 resolution processed by a 3x3 convolution kernel requires over five million operations in total. However, the kernel cannot exceed the

dimension of the image resolution. The convolution kernels (acquired from [145]) of the following fixed filters developed for the SARA prototype are depicted in Table A5-A8:

- Edge detection (Mexican hat/Marr) Filter: *reveals the edges of elements/objects of an image*
- Blur (Flat) Filter: *allows to adjust the softness of the focus and reduce the noise of an image*
- Edge detection (Laplacian) Filter: *reveals the edges of elements/objects of an image*
- Sharp Filter: *allows to sharpen the focus by increasing the contrast where colors or shades intersect*

Table A5. Edge detection (Mexican hat/Marr) Filter - 13x13 matrix

0.0	0.0	0.0	0.0	0.0	-1.0	-1.0	-1.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	-1.0	-1.0	-2.0	-2.0	-2.0	-1.0	-1.0	0.0	0.0	0.0
0.0	0.0	-2.0	-2.0	-3.0	-3.0	-4.0	-3.0	-3.0	-2.0	-2.0	0.0	0.0
0.0	-1.0	-2.0	-3.0	-3.0	-3.0	-2.0	-3.0	-3.0	-3.0	-2.0	-1.0	0.0
0.0	-1.0	-3.0	-3.0	-1.0	4.0	6.0	4.0	-1.0	-3.0	-3.0	-1.0	0.0
-1.0	-2.0	-3.0	-3.0	4.0	14.0	19.0	14.0	4.0	-3.0	-3.0	-2.0	-1.0
-1.0	-2.0	-4.0	-2.0	6.0	19.0	24.0	19.0	6.0	-2.0	-4.0	-2.0	-1.0
-1.0	-2.0	-3.0	-3.0	4.0	14.0	19.0	14.0	4.0	-3.0	-3.0	-2.0	-1.0
0.0	-1.0	-3.0	-3.0	-1.0	4.0	6.0	4.0	-1.0	-3.0	-3.0	-1.0	0.0
0.0	-1.0	-2.0	-3.0	-3.0	-3.0	-2.0	-3.0	-3.0	-3.0	-2.0	-1.0	0.0
0.0	0.0	-2.0	-2.0	-3.0	-3.0	-4.0	-3.0	-3.0	-2.0	-2.0	0.0	0.0
0.0	0.0	0.0	-1.0	-1.0	-2.0	-2.0	-2.0	-1.0	-1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	-1.0	-1.0	-1.0	0.0	0.0	0.0	0.0	0.0

Table A6. Edge detection (Laplacian) filter - 5x5 matrix

0.0	0.0	-1.0	0.0	0.0
0.0	-1.0	-2.0	-1.0	0.0
-1.0	-2.0	16.0	-2.0	-1.0
0.0	-1.0	-2.0	-1.0	0.0
0.0	0.0	-1.0	0.0	0.0

Table A7. Blur (lat) filter - 3x3 matrix

1.0/9.0	1.0/9.0	1.0/9.0
1.0/9.0	1.0/9.0	1.0/9.0
1.0/9.0	1.0/9.0	1.0/9.0

Table A8. Sharp filter - 3x3 matrix

0.0	-1.0	0.0
-1.0	5.0	-1.0
0.0	-1.0	0.0

A5. List of FIPA specification documents

Every FIPA specification document is referenced by a unique ID number, as assigned by FIPA[54]. This section provides a complete list of the FIPA specification documents referenced within this dissertation. The list contains the ID number of every specification document (in bold) and the title of the corresponding document along with its brief description. The letter enclosed in parenthesis after the title of each document identifies the status of the particular specification. FIPA specifications are not arbitrarily set, they have a life-cycle and in order for a specification to become standard two years of experimental tests must pass. Therefore, 'S' stands for a specification that has become a standard, whereas 'X' stands for a specification that is in the experimental phase. FIPA specification documents may be downloaded from FIPA web-site[54].

FIPA00001 - FIPA Abstract Architecture Specification (S)
An abstract agent architecture for FIPA.

FIPA00008 - FIPA SL Content Language Specification (S)
A description of the FIPA Semantic Language content language.

FIPA00009 - FIPA CCL Content Language Specification (X)
A description of the FIPA Constraint Choice Language content language.

FIPA00010 - FIPA KIF Content Language Specification (X)
A description of a FIPA content language based on the Knowledge Interchange Format.

FIPA00011 - FIPA RDF Content Language Specification (X)
A description of a FIPA content language based on the Resource Description Framework.

FIPA00014 - FIPA Nomadic Application Support Specification (S)
A description of agents and an ontology for supporting nomadic applications and devices.

FIPA00023 - FIPA Agent Management Specification (S)
Management for agents on FIPA agent platforms.

FIPA00026 - FIPA Request Interaction Protocol Specification (S)
The FIPA Request interaction protocol.

FIPA00027 - FIPA Query Interaction Protocol Specification (S)
The FIPA Query interaction protocol.

FIPA00028 - FIPA Request When Interaction Protocol Specification (S)
The FIPA Request When interaction protocol.

FIPA00029 - FIPA Contract Net Interaction Protocol Specification (S)
The FIPA Contract Net interaction protocol.

FIPA00030 - FIPA Iterated Contract Net Interaction Protocol Specification (S)
The FIPA Iterated Contract Net interaction protocol.

FIPA00031 - FIPA English Auction Interaction Protocol Specification (X)
The FIPA English Auction interaction protocol.

FIPA00032 - FIPA Dutch Auction Interaction Protocol Specification (X)
The FIPA Dutch Auction interaction protocol.

FIPA00033 - FIPA Brokering Interaction Protocol Specification (S)
The FIPA Brokering interaction protocol.

FIPA00034 - FIPA Recruiting Interaction Protocol Specification (S)
The FIPA Recruiting interaction protocol.

FIPA00035 - FIPA Subscribe Interaction Protocol Specification (S)
The FIPA Subscribe interaction protocol.

FIPA00036 - FIPA Propose Interaction Protocol Specification (S)
The FIPA Propose interaction protocol.

FIPA00037 - FIPA Communicative Act Library Specification (S)
A library of FIPA communicative acts and requirements for new communicative acts.

FIPA00061 - FIPA ACL Message Structure Specification (S)
A description of the structure of FIPA ACL.

FIPA00067 - FIPA Agent Message Transport Service Specification (S)
A description of the Message Transport Service for agents on FIPA agent platforms.

FIPA00069 - FIPA ACL Message Representation in Bit-Efficient Specification (S)
A description of an ACL message representation in a bit-efficient encoding.

FIPA00070 - FIPA ACL Message Representation in String Specification (S)
A description of an ACL message representation in a string encoding.

FIPA00071 - FIPA ACL Message Representation in XML Specification (S)
A description of an ACL message representation in an XML encoding.

FIPA00075 - FIPA Agent Message Transport Protocol for IIOP Specification (S)
A description of a message transport protocol based on IIOP.

FIPA00076 - FIPA Agent Message Transport Protocol for WAP Specification (X)
A description of a message transport protocol based on WAP.

FIPA00080 - FIPA Personal Travel Assistance Specification (X)
A description of agents and an ontology for supporting personal travel assistance applications.

FIPA00081 - FIPA Audio-Visual Entertainment and Broadcasting Specification (X)
A description of agents and an ontology for supporting entertainment and broadcasting applications.

FIPA00082 - FIPA Network Management and Provisioning Specification (X)
A description of agents and an ontology for network management and provisioning applications.

FIPA00083 - FIPA Personal Assistant Specification (X)
A description of agents and an ontology for personal assistant applications.

FIPA00084 - FIPA Agent Message Transport Protocol for HTTP Specification (S)
A description of a message transport protocol based on HTTP.

FIPA00085 - FIPA Agent Message Transport Envelope Representation in XML Specification (S)

A description of a message transport envelope representation in an XML encoding.

FIPA00088 - FIPA Agent Message Transport Envelope Representation in Bit Efficient Specification (S)

A description of a message transport envelope representation in a bit efficient encoding.

References

[1] AgentCities - a global, collaborative effort to construct an open network of on-line systems hosting diverse agent based services, http://www.agentcities.org, *(last visited 2005)*.

[2] AgentCities node hosted by UMIST (University of Manchester Institute of Science and Technology) in UK, http://www.agentcities.co.umist.ac.uk, *(last visited 2005)*.

[3] Aglets - agent platform developed by IBM, http://www.research.ibm.com/trl/aglets/index.html, *(last visited 2005)*.

[4] Aloisio G., Cafaro M., Williams R. "The Digital Puglia Project: An Active Digital LIbrary of Remote Sensing Data". In Proceedings of the 7th International Conference on High Performance Computing and Networking Europe, April 12 - 14, 1999, Springer Lecture Notes in Computer Science vol. 1593, Amsterdam, The Netherlands, 1999.

[5] Aloisio G., Milillo G., Williams R.D. "An XML Architecture of high-performance web-based analysis of remote-sensing archives". In Future Generation Computer Systems, vol 16, pp 91-100, 1999.

[6] Altmann J., Gruber F., Klug L., Stockner W., Weippl E. "Using mobile agents in the real world: a survey and evaluation of agent platforms". In Proceedings of the 2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS at the 5th Int. conference on Autonomous Agents, ACM Press, 2001.

[7] AMPEX - a provider of data storage and access management systems, http://www.ampexdata.com, *(last visited 2005)*.

[8] Andresen D., Carver L., Dolin R., Fischer C., Frew J., Goodchild M., Ibarra O., Kothuri R., Larsgaard M., Manjunath B.S., Nebert D., Simpson J., Smith T.R., Yang T., Zheng Q. "The WWW prototype of the Alexandria digital library". In Proceedings of ISDL'95 (Int. Symp. on Digital Libraries), pp.17-27, Japan, 1995.

[9] Artsy Y., Finkel R. "Designing a process migration facility: the Charlotte experience". Computer, 22(9):47-56, 1989

[10] Austin J.L. "How to do things with words", Oxford University Press, 1962.

[11] Backschat M., Pfaffinger A., Zenger C. "Economic-based dynamic load distribution in large workstation networks". In Proceedings of the 2nd Int. Euro-Par Conference, volume 2, Lyon, France, pp. 631-634, 1996.

[12] Baldeschwieler J.E., Blumofe R.D., Brewer E.A. "ATLAS: an infrastructure for global computing". In Proceedings of the 7th ACM SIGOPS European Workshop: Systems Support for Worldwide Applications, 1996

[13] Bernd C.O., Cappello P., Ionescu M.F., Neary M.O., Schauser K.E., Wu D. "Javelin: internet-based parallel computing using Java". In Workshop on Java for Science and Engineering Computation, ACM press, Las Vegas, 1997.

[14] Birmingham W., Durfee E., Mullen T., Wellman M. "The distributed agent architecture of the University of Michigan digital library". IEEE Computer, special issue on Building Large-scale Digital Libraries, 1996.

[15] Bond A.H., Gasser L. "Readings in distributed artificial intelligence". Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[16] Bishop A.P. "Digital libraries and knowledge disaggregation: the use of journal article components". In Proceedings of the ACM Digital Libraries 1998 Conference, New York, 1998.

[17] Braun P., Erfurth C., Rossak W. "Performance evaluation of various migration strategies for mobile agents". In Proceedings of Kommunikation in Verteilten Systemen (KiVS), Hamburg, Germany, Springer, ISBN 3-540-41645-5, 2001.

[18] Brazier F.M.T., Overeinder B.J., Steen V.M., Wijngaards N.J.E. "Agent factory: generative migration of mobile agents in heterogeneous environments". In Proceedings of the 2002 ACM Symp. on Applied Computing (SAC 2002), pp.101-106, 2002.

[19] Bredin J., Kotz D., Rus D. "Market-based resource control for mobile agents". In Proceedings of the 2nd Int. conference on Autonomous Agents (AA98), pp 197-204, Mineapolis, USA, ACM press, 1998.

[20] Brogden W.B., Louie J.A., Tittel E. "Visual Café for Java database development edition". 1st edition, Coriolis group Inc., ISBN 1-57610-219-X, 1998.

[21] Broos R., Dillenseger B., Dini P., Hong T., Leichsenring A., Leith M., Malville E., Nietfeld M., Sadi K., Zell M. "Mobile Agent Platform Assessment Report". Contribution to the EU Advanced Communications Technology and Services (ACTS) Programme, 2000.

[22] BTexact - ISR agent research concerned with the development and analysis of sophisticated AI problem-solving and control architectures for both single-agent and multiple-agent systems (MATS project), http://more.btexact.com/projects/agents.htm, *(last visited 2005)*.

[23] Burg B., Dale J., Willmott S. "Open standards and open sources for agent-based systems". Article in Agentlink, news 6, 2001.

[24] CAMELEON (Communication Agents for Mobility Enhancements in a Logical Environment of Open Networks), ACTS, Project AC341, http://www.comnets.rwth-aachen.de/~cameleon/, *(last visited 2005)*.

[25] CAVE (Automatic Virtual Environment), http://www.fakespacesystems.com/pdfs/051603/CAVE.pdf, *(last visited 2005)*.

[26] Charles K.N., Grindell J.M. "Interoperating java mobile agents". Published in special issue on Mobile Agents, Walter Binder, ed., Informatik Forum, July 2002.

[27] Charlton P., Bonnefoy D., Lhuillier N. "Dealing with interoperability for agent based services". In Proceedings of the 5th Int. Conference on Autonomous Agents, ACM press, ISBN:1-58113-326-X, pp.236-237, Montreal, Quebec, Canada, 2001.

[28] Chavez A., Moukas A., Maes P. "Challenger: a multi-agent system for distributed resource allocation". In Proceedings of the 1st Int. Conference on Autonomous Agents (AA97), ACM Press, Marina del Ray, CA, USA, 1997.

[29] Chess D., Grosof B., Harrison C., Levine D., Parris C., Tsudik G. "Itinerant agents for mobile". IEEE Personal Communications, vol. 2, No. 5, 1993.

[30] Christel M., Kanade T., Mauldin M., Reddy R., Stevens S., Wactlar H. "Techniques for the creation and exploration of digital video libraries, multimedia tools and applications". Volume 2, Borko Furht, editor, MA: Kluwer Academic Publishers, Boston, 1996.

[31] Christel M., Martin D. "Information visualization within a digital video library". Journal of Intelligent Information Systems 11(3), pp. 235-257, 1998.

[32] Coddington P.D., Hawick K.A., James H.A. "Web-based access to distributed high-performance geographic information systems for decision support". In Proceedings of HICSS-32, Maui, 1999

[33] Coddington P.D., Hawick K.A., Kerry K.E., Mathew J.A., Silis A.J., Webb D.L., Whitbread P.J., Irving C.G., Grigg M.W., Jana R., Tang K. "Implementation of a geospatial imagery digital library using Java and CORBA". In Proceedings of Technologies of Object-Oriented Languages and Systems Asia '98 (TOOLS 27), Beijing, 1998.

[34] Cogan P., Gomoluch J., Schroeder M. "A quantitative and qualitative comparison of distributed information processing using mobile agents realised in RMI and Voyager". Journal of Software Engineering and Knowledge Engineering, 11(5):583-605, World Scientific, 2002.

[35] CORBA (Common Object Request Broker Architecture), http://www.corba.com, *(last visited 2005).*

[36] Corradi A., Leonardi L., Zambonelli F. "On the effectiveness of different diffusive load balancing policies in dynamic applications". Conference on High-Performance Computing and Networking (HPCN-98), Lecture Notes in Computer Science, No. 1401, Springer-Verlag (D), April 1998.

[37] Cortes A., Ripoll A., Senar M.A., Luque E. "Performance comparison of dynamic load-balancing strategies for distributed computing". In Proceedings of the 32nd Hawaii Int. Conference on System Sciences, IEEE, January 1999.

[38] Crum L. "University of Michigan digital library project", Communications of the ACM 38, pp. 63-64, April 1995.

[39] Dale J., Mamdani E. "Open standards for interoperating agent-based systems". In Software Focus, 1(2), Wiley, 2001.

[40] Data explorer - a visualisation tool, http://www.opendx.org, *(last visited 2005).*

[41] Davidson S., Overton C., Bunerman P. "Challenges in integrating biological data sources". Journal in Computational biology, 1995

[42] Decker K., Sycara K., Williamson M. "Cloning for intelligent adaptive information agents". In Multi-Agent Systems: Methodologies and Applications, Lecture notes in Artificial intelligence 1286, Springer, pp.63-75, 1997.

[43] Dikaiakos M., Samaras G. "Quantitative performance analysis of mobile agent systems: a hierarchical approach". Technical Report TR-00-2, Dept. of Computer Science, University of Cyprus, Nicosia, Cyprus, June 2000.

[44] Distributed.net - an organization which encompass thousands of users around the world, resulting in a parallel computing power, http://www.distributed.net, *(last visited 2005).*

[45] Document Content Description (DCD) for XML, http://www.w3.org/TR/NOTE-dcd, *(last visited 2005).*

[46] Douglas F., Ousterhout J. "Transparent process migration: design alternatives and the sprite implementation". Software Practice and Experience, 21(8):757-85, 1991.

[47] Durfee E.H., Lesser V. "Negotiating task decomposition and allocation using partial global planning". In L. Gasser and M. Huhns, editors, Distributed Artificial Intelligence volume II, pp. 229-244, Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989.

[48] Eager D.L., Lazowska E.D., Zahorjan J. "Adaptive load sharing in homogeneous distributed systems". IEEE Trans on Software Engineering, vol SE-12, pp. 662-675, 1986.

[49] Erfurth C., Braun P., Rossak W. "Migration intelligence for mobile agents". In Proceedings of Artificial Intelligence and the Simulation of Behaviour (AISB) Symp. on Software mobility and adaptive behaviour. University of York, United Kingdom, pp. 81-88, 21st-24th March 2001.

[50] E-mail communication with Dr. Holger in 2002 - a member of the FLASH project, http://www.iti.uni-luebeck.de/Research/PC/Flash/contact.php3, *(last visited 2005)*, 2002.

[51] FACTS (FIPA Agent Communication Technologies and Services), ACTS Project AC317., http://www.labs.bt.com/profsoc/facts/, *(last visited 2005)*.

[52] FileTek - a provider of data storage and access management systems, http://www.filetek.com, *(last visited 2005)*.

[53] Finin T., Labrou Y., Mayfield J. "KQML as an agent communication language". In software agents, J.M. Bradshaw (ed), MIT Press, Cambridge, Mass, pp. 291-316, 1997.

[54] FIPA (Foundation of Intelligent Physical Agents), http://www.fipa.org, *(last visited 2005)*.

[55] FIPA Communicative Act Library Specification, http://www.fipa.org/specs/fipa00037/, *(last visited 2005)*.

[56] FIPA-compliant agent-based applications, http://www.fipa.org/resources/byproject.html, *(last visited 2005)*.

[57] FIPA-compliant publicly available agent platform implementations, http://www.fipa.org/resources/livesystems.html#os, *(last visited 2005)*.

[58] FIPA inform - The newsletter of the Foundation for Intelligent Physical Agents, Vol. 3, Issue 4, January 2003, http://www.fipa.org/docs/output/f-out-00137/, *(last visited 2005)*, Jan 2003.

[59] FIPA-NET (discontinued project) - details can be found in "Inform: the newsletter for the foundation for intelligent physical agents, Issue 2", July 2002, http://fipa.umbc.edu/inform/inform2.pdf, *(last visited 2005)*, Jul 2002.

[60] FIPA-OS component-based toolkit, http://www.nortelnetworks.com/products/announcements/fipa/info.html, *(last visited 2005).*

[61] FIPA-OS Inter-platform Communications Configuration Guide, p.15, http http://fipa-os.sourceforge.net/ docs/Interplatform_Configuration_Guide.pdf , *(last visited 2005)*, March 2002.

[62] FIPA Security SIG Request for Informarion, http://www.fipa.org/docs/output/f-out-00065/, *(last visited 2005)*, Jun 2000.

[63] FIPA97 Agent Management, Document OC00019, Version 2.0 Part 1, http://www.fipa.org/specs/ fipa00019/OC00019.pdf, *(last visited 2005)*, Oct 1998.

[64] FIPA98 Agent Security, Document OC00020, Version 1.0 Part 1.0, http://www.fipa.org/specs/ fipa00020/OC00020.pdf, *(last visited 2005)*, Oct 1998.

[65] FLASH (Flexible Agent System for Heterogeneous Cluster) - an agent-based framework for the creation of load-balanced distributed applications running on a heterogeneous cluster systems, http://www.iti.mu-luebeck.de/Research/PC/Flash/, *(last visited 2005).*

[66] Fox E.A., "Digital Libraries". Hot topics, IEEE Computer 26(11), pp.79-81, November 1993.

[67] Frew J., Freeston M., Freitas N., Hill L., Janee G., Lovette K., Nideffer R., Smith T., Zheng Q., Nikolaou C., Stephanidis C. "The Alexandria digital library architecture". In Proceedings of the 2nd European Conference on Research and Advanced Technology for Digital Libraries (ECDL'98), pp 61-73, Crete, Greece, 1998.

[68] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V. "PVM: Parallel Virtual Machine". MIT press, Cambridge, 1994

[69] Ghanea-Hercock R., Collis J.C., Ndumu D.T. "Co-operating mobile agents for distributed parallel processing". In Proceedings of the 3rd Int. Conference on Autonomous Agents (AA99), ACM press, Mineapolis, USA, 1999.

[70] Gomoluch J., Schroeder M. "Information agents on the move: A survey on load-balancing with mobile agents". Software Focus, vol. 2, no. 2, Wiley, 2001.

[71] Gonne M., Grewe C., Pals H. "Monitoring of Mobile Agents in Large Cluster Systems". Published in IEEE Int. Symp. on Network Computing and Applications, 2001.

[72] Grasshopper - agent platform developed by IKV++, http://www.grasshopper.de/, *(last visited 2005)*.

[73] Gray P.A., Sunderam V.S. "The IceT environment for parallel and distributed computing". In Proceedings of ACM Workshop on Java for Science and Engineering Computation, pp. 275-282, 1997.

[74] Green S., Hurst L., Nangle B., Cunningham P., Somers F., Evans R., "Software agents: A Review". Technical Document TCD-CS-97-04, Computer Science Department, Trinity College Dublin, May 1997.

[75] Gruber T.R. "A translation approach to portable ontology specifications". In Knowledge Acquisition, 5(2):199-220, Academic press 1993.

[76] Hagg S. "A sentinel approach to fault handling in multi-agent systems". In Proceedings of the 2nd Australian Workshop on Distributed AI, Cairns, Australia, 1997.

[77] Harrison C.G., Chessm D.M., Kershenbaum A. "Mobile agents: are they a good idea?". Research report, IBM Research, 1995.

[78] Hatchol-Balter M. "Task assignment with unknown duration". In Proceedings of Int. Conference on Distributed Computing Systems, journal of the ACM (JACM) ISSN:0004-5411, vol.49, issue 2, pp.260-288, 2002.

[79] Hatchol-Balter M., Crovella M., Murta C. "On choosing a task assignment policy for a distributed server system". IEEE journal of Parallel and Distributed Computing, vol.59, pp.204-228, 1999.

[80] Hirano S., Yasu Y., Igarashi H. "Performance evaluation of popular distributed object technologies for Java". Concurrency: Practice and Experience 10(11-13), pp.927-940, 1998.

[81] Holger P. MSc thesis on: Lastverwaltung in workstation-clustern auf basis von mobilen agenten (Load administration in workstation-cluster on basis of mobile agents). Submitted in the University of Luebeck, Dept. of institute of computer engineering, in Germany, 2000.

[82] Holger P., Claus G. "Dynamisch-adaptive lastverwaltung für mobile agenten" (Dynamic-adaptive load administration for mobile agents). In Proceedings of the 19th PARS Workshop (Basel), Mitteilungen - Gesellschaft für Informatik e.V. Parallele Algorithmen und Rechnerstrukturen, Nr. 20, 97-108, Gesellschaft für Informatik e.V., Bonn 2003.

[83] ImmersaDesk - a 3D visualisation tool, http://www.pdc.kth.se/compresc/machines/idesk.html, *(last visited 2005).*

[84] IRIS explorer - a visualisation tool, http://www.nag.co.uk/Welcome_IEC.html, *(last visited 2005).*

[85] JADE (Java Agent DEvelopment Framework) - agent platfrom developed by TILAB and AOT, http://sharon.cselt.it/projects/jade, *(last visited 2005).*

[86] JAI (Java Advanced Imaging) Java Development Kit, http://java.sun.com/products/java-media/jai, *(last visited 2005).*

[87] JAS (Java Agent Services), http://www.jcp.org/en/jsr/detail?id=87, *(last visited 2005).*

[88] James H.A., Hawick K.A. "A web-based interface for on-demand processing of satellite imagery archives". In Australian Computer Science Communications, vol.20, Springer-Verlag Pte Ltd, 1998.

[89] Java - programming language, http://java.sun.com/, *(last visited 2005).*

[90] Java Servlet Technology, http://java.sun.com/products/servlet/index.html, *(last visited 2005).*

[91] Jennings N.R., Sycara K., Wooldridge M. "A roadmap of agent research and development". Int. journal of Autonomous Agents and Multi-Agent Systems, 1998

[92] Johansen D., Renesse V.R., Schneider F.B. "Operating system support for mobile agents". In Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems, pp.42-45, May 1995.

[93] Jul E., Levy H., Hutchinson N., Black A. "Fine-grained mobility in the emerland system". In Proceedings of 11th ACM Symp. on Operating systems principles, ISBN: 0-89791-242-X, pp.105-106, Austin, Texas, United States, 1987.

[94] Kambil A. "Different auction models", http://pages.stern.nyu.edu/~akambil/teaching/cases/auction/appendix.html, *(last visited 2005)*.

[95] Keren A., Barak A. "Adaptive placement of parallel java agents in a scalable computing cluster". In Proceedings of the Workshop on Java for High Performance Network Computing, ACM Press, Stanford University, Palo Alto, CA, USA, 1998.

[96] KQML (Knowledge Query Meta Language), http://www.cs.umbc.edu/kqml/, *(last visited 2005)*.

[97] Magnin L., Snoussi H., Pham V.T., Dury A., Nie J.Y. "Agents need to become welcome". In Proceedings of the 3rd Int. Symp. on Multi-Agent Systems, Large Complex Systems, and E-Businesses (MALCEB'2002), Erfurt/Thuringia, Germany, October 2002.

[98] Malone T.W., Fikes R.E., Grant K.R., Howard M.T. "Enterprise: a market-like task scheduler for distributed computing environments". In the Ecology of Computation. Ed. Huberman, B. A. Elsevier, Holland, 1988.

[99] Marchionini, G. "Digital Library Research and Development". In A. Kent (Ed.) Encyclopedia of Library and Information Science, vol. 63, supplement 26, NY:Marcel Dekker, pp. 259-279, 1998.

[100] Marin O., Sens P., Briot J.P., Guessoum Z. "Towards adaptive fault tolerance for distributed multi-agent systems". In Proceedings of 3rd European Research Seminar on Advanced Distributed Systems (ERSAD'2001), pp.195-201, Bertinoro, Italy, 2001.

[101] MASIF - The Object Management Group's Mobile Agent System Interoperability Facility, http://www.omg.org, *(last visited 2005)*.

[102] Mills D. L. "Network Time protocol (version 3) Specification, Implementation and Analysis", Network Working Group RFC-1305, University of Delaware, *(last visited 2005)*, 1992.

[103] Minsky Y., Renesse V.R., Schneider F.B., Stoller S.D. "Cryptographic support for fault-tolerant distributed computing". In Proceedings on 7th ACM SIGOPS European Workshop, pp.109-114, ACM Press, Connemara, Ireland, September 1996.

[104] Misikangas P., Raatikainen K. "Agent migration between incompatible platforms". In the 20th Int. Conference on Distributed Computing Systems (ICDCS 2000), Taipei, Taiwan, Republic of China, April 2000.

[105] Mole - agent platform developed at the University of Stuttgart, http://mole.informatik.uni-stuttgart.de/, *(last visited 2005)*.

[106] Nabil R.A., Vijayalakshmi A., Igg A. "SI in digital libraries". Communications of the ACM, vol.43, no.6, pp.64-72, June 2000.

[107] Obeloeer W., Grewe C. "Load management with mobile agents". In Proceedings of the 24th EUROMICRO Conference, IEEE Computers, pp.1005-1012, 1998.

[108] Object Management Group - Agent Technology Green paper, OMG Document ec/2000-08-01 version 1.0, http://www.jeffsutherland.com/papers/OMG/Green_Paper_v080.html, *(last visited 2005)*, August 1999.

[109] OCEAN (Open Computation Exchange & Auctioning (or Arbitration) Network), commercial buying and selling of dynamic distributed computing resources over the internet, http://www.cise.ufl.edu/~mpf/ ocean/index.htm, *(last visited 2005)*.

[110] OMG Request For Information, Agent Technology in OMA, OMG Document #ec/99-03-10, http://www.objs.com/isig/agent-rfi-6.html, *(last visited 2005)*, August 1999.

[111] Oracle Data-Base Management System, http://www.oracle.com, *(last visited 2005)*.

[112] Oreilly Inc. servlet JDK, http://www.stanford.edu/group/coursework/stanfordoki/oreilly/, *(last visited 2005)*.

[113] Othman O., O'Ryan C., Schmidt D. "Strategies for CORBA Middleware-Based Load Balancing". IEEE DS Online, 2(3), April 2001.

[114] Pacheco P.S. "Parallel computing with MPI". Morgan Kaufmann, Publishers Inc. ACM press, ISBN: 1-55860-339-5, San Francisco, CA, USA, 1996.

[115] Paepcke A., Baldonado M., Chang C., Cousins S., Garcia-Molina H. "Using distributed objects to build the Stanford digital library Infobus". IEEE Computer, 1999.

[116] Pals H., Petri S., Grewe C. "FANTOMAS: Fault Tolerance for Mobile Agents in Clusters". In Proceedings of 15th IPDPS 2000 Workshops, Cancun, Mexico, May 2000, Lecture Notes in Computer Science (LNCS) 1800, pp. 1236-1247, Springer-Verlag, Berlin 2000.

[117] Panti M., Penserini L., Spalazzi L., Valenti S. "A FIPA compliant agent platform for federated information systems". In ACIS Int. Journal of Computer & Information Science, volume 1, issue 3. Special issue on software engineering applied to networking & parallel/distributed computing, ISSN:1525-9293, pp. 145-156, USA, 2000.

[118] Parent C., Spaccapietra S. "Database integration: an overview of issues and approaches". Communications of the ACM, vol. 41, no 5, pp. 166-178, 1998.

[119] Poslad S., Calisti M. "Towards improved trust and security in FIPA agent platforms". In Proceedings of Autonomous Agents 2000 Workshop on Deception, Fraud and Trust in Agent Societies, Spain, 2000.

[120] Poslad S., Buckle P., Hadingham R. "Open source standards and scaleable agencies". Presented at the Autonomous Agents 2000 Workshop on Infrastructure for Scalable Multi-agent Systems, Spain, 2000.

[121] Poslad S., Buckle P., Hadingham R. "The FIPA-OS agent platform: open source for open Standards". In Proceedings of PAAM 2000, Manchester, UK, 2000.

[122] Rao S., Alvisi L., Vin H.M. "Hybrid message logging protocols for fast recovery". In Digest of Fast Abstracts: FTCS-28 - The 28th annual Int. Symp. on fault-tolerant computing, IEEE Computer Society, pp. 41-42, June 1998.

[123] Rasmusson A., Olsson T., Hansen P. "A virtual community library: SICS digital library infrastructure project". Research and advanced Technology for Digital Libraries. Second European Conference, ECDL'98, Greece-Crete, 1998.

[124] RSI - Recursion Software Inc., Voyager Package, http://www.recursionsw.com/, *(last visited 2005).*

[125] Resource Description Framework (RDF) schema specification, http://www.w3.org/TR/WD-rdf-schema, *(last visited 2005)*, Feb 2004.

[126] Samaras G., Dikaiakos M., Spyrou C., Liverdos A. "Mobile agent platforms for web databases: a qualitive and quantitative assesment". In Proceedings of ASAMA'99: IEEE-Computer Society, pp. 50-64, October, 1999.

[127] Sadek M. D., Bretier P., Cadoret V., Cozannet A., Dupont P., Ferrieux A. Panaget F. "A Co-operative Spoken Dialogue System Based Upon a Rational Agent Model: A First Implementation of the AGS Application". In Proceedings of the ESCA/ETR Workshop on SpokenDialogueSystems: Theories and Applications, Denmark, 1995.

[128] Sandholm T. "Distributed rational decision making". In the textbook Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, Weiss, G. (ed.), MIT press, 1999.

[129] SARA - The Synthetic Aperture Radar Atlas, online Digital Library, http://sara.unile.it/sara/, *(last visited 2005).*

[130] Sarmenta L.F.G., Hirano S. Bayanihan "Building and studying web-based volunteer computing using Java". In Future generation computer systems, 15 (5/6), 1999.

[131] Schroeder B., Harchol-Balter M. "Evaluation of task assignment policies for supercomputing servers: the case for load unbalancing and fairness". In Proceedings of the 9th IEEE Int. Symp. on High Performance Distributed Computing (HPDC'00), ACM press, ISBN:0-7695-0783-2, 2000.

[132] Scherson I.D., Campos L.M. "A distributed dynamic load balancing strategy based on rate of change". In Parallel Computing Workshop, Singapore, pp. P2-I 2-8.September 1998.

[133] Searle J.R. Speech acts, Cambridge University Press, 1969.

[134] Shaikh A., Rexford J., Shin K.G. "Load-sensitive routing of long-lived IP flows". In Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication, ACM press, ISBN:1-58113-135-6, Vol. 29, issue 4, pp. 215 - 226, 1999.

[135] Shen W., Norrie D.H. "A hybrid agent-oriented infrastructure for modeling manufacturing enterprises". In Proceedings of KAW'98, Agent-5:1-19,Banff, Canada, 1998.

[136] Shoch F., Hupp J. "The worm programs - early experience with a distributed computation". Communications of the ACM, 25(3):172-80, 1982.

[137] Silva L., Soares G, Martins P., Batista V., Santos L. "Comparing the performance of mobile agent systems: a study of benchmarking". Computer Communications 23(8), pp. 769-778, 2000

[138] Silva P.S., Mendes M.J. "A framework for adding computational intelligence to mobile agents". Symp. on software mobility and adaptive behaviour, AISB'01, UK, 2001.

[139] Special Issue on Digital Libraries. Communications of the ACM 38(4), New York, April 1995.

[140] Special Issue on Digital Libraries, IEEE Computer Magazine, May 1996.

[141] SSL (Secure Sockets Layer), http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci343029,00.html, *(last visited 2005).*

[142] Strasser M., Rothermel K. "Reliability concepts for mobile agents". Int. journal of cooperative information systems (IJCIS), 7(4):355-382, 1998.

[143] Straoer M., Schwehm M. "A Performance Model for Mobile Agent Systems". In Proceedings of the Int. Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'97, Vol. 2, pp. 1132-1140, Las Vegas NV, 1997.

[144] Stroud K.A. "Further engineering mathematics", 3rd edition, pp. 125-170 and 391-451, published by PALGRAVE, ISBN 0-333-65741-1, 1996.

[145] Sun Microsystems. "Programming in Java Advanced Imaging", release 1.0, http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/, *(last visited 2005)*, July 1999.

[146] Tanenbaum A.S. "Modern operating systems". Englewood Cliffs, New Jersey: Prentice-Hall, 1992

[147] The Distributed-Parallel Storage System (DPSS) - http://www-didc.lbl.gov/DPSS, *(last visited 2005)*.

[148] The European ACTS research program - "CAMELEON", Performance Assessment Results (Final Version) Deliverable D09, http://www.comnets.rwth-aachen.de/~cameleon/, *(last visited 2005)*.

[149] The High Performance Storage System - http://www4.clearlake.ibm.com/hpss/index.jsp, *(last visited 2005)*.

[150] The NASA/JPL Imaging Radar, http://southport.jpl.nasa.gov/, *(last visited 2005)*.

[151] University of Edinburgh, Heriot-Watt - department of mathematics and computer science, http://www.macs.hw.ac.uk, *(last visited 2005)*.

[152] Vinoski S. "CORBA: integrating diverse applications within distributed heterogeneous environments". In IEEE Communications magazine, 1997.

[153] Visualisation and Processing of Multichannel Images – California Institute of Technology, Center of Advanced Computing Research, http://www.carc.caltech.edu/SDA/DigiPuglia/multichannel.htm, *(last visited 2005)*.

[154] Voyager agent-platform, Recursion Software Inc., http://www.recursionsw.com/osi.asp, *(last visited 2005)*.

[155] Voyager Ease of Development whitepaper, from Recursion Software Inc., www.recursionsw.com/Voyager/Ease_of_Development.pdf, *(last visited 2005)*, Jan 2002.

[156] Voyager ORB developer guide, Recursion Software Inc., http://www.recursionsw.com/Voyager/Voyager_User_Guide.pdf, *(last visited 2005)*, 2003.

[157] Waldspurger C.A., Hogg T., Huberman B.A., Kephart J.O., Stornetta W.S. "Spawn: a distributed computational economy". IEEE Transactions on Software Engineering 18(2), pp. 103-117, 1992.

[158] Walsh T., Paciorek N., Wong D. "Security and Reliability in Concordia". Published in Mobility: Processes, Computers and Agents, Addison Wesley, ISBN 0-201-37928-7, pp. 524-534, April 1999.

[159] Wilensky R. UC Berkeley's Digital Library project, Communications of the ACM 38, p.60, April 1995.

[160] Williams R.D., Bunn J., Moore R. "Interfaces to scientific data archives". Workshop on interfaces to scientific data archives, Pasadena, California, 1998.

[161] Williams R.D., Sears B. "A high-performance active digital library". Parallel computing, special issue in Metacomputing, Nov 1998.

[162] Williams R.D., Sears B. "A web-based interface for on-demand processing of satellite imagery archives". In Parallel computing, special issue on metacomputing, 1998.

[163] Wims B., Xu C.Z. "Traveler: a mobile agent infrastructure for wide area parallel computing". In Proceedings of the IEEE Joint Sumposium ASA/MA'99: 1st Int. Symp. on Agent Systems and Applications (ASA'99) and 3rd Int. Symp. on Mobile Agents (MA'99), Palm Springs, 1999.

[164] XML - eXtensible Markup Language, http://www.xml.org, *(last visited 2005)*.

[165] XML-Data, http://www.w3.org/TR/1998/NOTE-XML-data, *(last visited 2005)*, Jan 1998.

[166] Xu C.Z., Tschoke S., Monien B. "Performance evaluation of load distribution strategies in parallel branch and bound computations". In Proceedings of the 7th Symp. on Parallel and Distributed Processing (SPDP'95), IEEE Computer Society Press, pp. 402-405, 1995.

[167] Xu C.Z., Wims B. "A mobile agent based push methodology for global parallel computing". In Proceedings of the 1st Int. Symp. on Agent Systems and Applications (ASA'99) and 3rd Int. Symp. on Mobile Agents (MA'99). IEEE, 1999.

[168] X.509 Digital certification - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcecryp2/ html/ ceconx509digitalcertification.asp, *(last visited 2005).*

[169] Zambonelli F. "How to improve local load balancing policies by distorting load information". In Proceedings of the 5th Int. Conference on High-Performance Computing, IEEE CS Press, Madras (IN), December 1998.

[170] Zeus - agent platform developed by British Telecommunications, http://more.btexact.com/projects/ agents/ zeus/, *(last visited 2005).*

[171] BACnet Conference & Expo: XML Takes Center Stage, http://www.hpac.com/member/feature/2004/0402/ 0402gipson.htm, *(last visited 2005).*

[172] eMule - File sharing system, http://www.emule-project.net/home/perl/general.cgi?l=1, *(last visited 2005).*

[173] eDonkey - File sahring system, http://www.edonkey2000.com/, *(last visited 2005).*

[174] Morpheus - File sharing system, http://morpheus.com/, *(last visited 2005).*

[175] Napster - File sharing system, http://www.napster.com/, *(last visited 2005).*

[176] ESA - European Space Agency, http://www.esa.int/esaEO/SEM913VZJND_index_0.html, *(last visited 2005).*