Chapter

1

1. Introduction

1.1. Introduction

"Don't keep forever on the public road, going only where others have gone. Leave the beaten track occasionally….. You will be certain to find something you have never seen before. Of course, it will be a little thing, but don't ignore it. Follow it up, explore all around it; one discovery will lead to another, and before you know it you will have something worth thinking about to occupy your mind. All really big discoveries are the results of thoughts." - Alexander Graham Bell

The latest trends and developments in computer sciences have brought a new revolution in the world. Computer is equipment, which has its application in every aspect of life. It has made the world a global village where data and inferences can be shared with others very conveniently. This data could be a nuclear research or just a prank. This present work deals with handling of data on a distributed Grid like network.

1.2. Grid - Overview

As far as technology goes, the Internet is the state of the art for many people in the world. It has facilitated innovations that make possible new services, empower employees, consumers and citizens with access to the latest information, allowing organization to cement new forms of strategic partnerships and enables new forms of sharing.

It is now time for the next evolutionary step to raise the bar for the positive benefits of communication; this step increasingly appears to be related to Grid computing. Grid

computing makes possible a phenomenon that is beyond enhanced collaborative communications or the sharing of information, it allows for communities to share actual computing resources as they tackle common goals. These communities can link their data, computers, sensors and other resources into a single virtual environment. Every resource can be wrapped into a service that is accessible across the hard boundaries of geography and the soft borders of institutions.

The computational power grid is analogous to electric power grid. Grid computing allows to couple geographically distributed resources and offers consistent and inexpensive access to resources irrespective of their physical location or access point. It enables sharing, selection, and aggregation of a wide variety of geographically distributed computational resources (such as supercomputers, compute clusters, storage systems, data sources, instruments, people). Thus allowing them to be used a single, unified resource for solving large-scale compute and data intensive computing applications (e.g, molecular modeling for drug design). Therefore a Computational Grid is a collection of heterogeneous computers and resources spread across multiple administrative domains with the intent of providing users uniform access to these resources. In simple words a Grid is a virtual distributed super computer.

Ultimately, the Grid will open up storage and transaction powers the same way that the Web has opened up access to content. Computing will become a utility just as any other utility, and will be ubiquitously accessible. This can break the desktop prison and individual users could use hand-held devices, mobile phones, public access points or other as yet undiscovered mechanisms to access computing resources and services that could exist anywhere in the world.

Grid related research and projects. The Grid is an active research area that is rapidly taking shape through the efforts of leading scientists across the world. It will provide access to computing resources, databases, hard drive storage, sensors, input devices, people and information stores in a pervasive, dependant, consistent and inexpensive manner. This can have a dramatically transforming effect on the range of applications that are possible; a sensational impact on human capabilities and society is expected.

As of date, the Grid as envisaged does not exist as yet, but it is on its way. The building blocks of the Grid are readily available in the form of networking technologies, physical hardware, access and security policies, algorithms and techniques for applications as varied as nuclear engineering to electronic commerce. The Grid will ultimately take

shape as the various network communication, security and resource brokering challenges are surmounted. We will one day be able to access resources across organizations that are autonomous and independent, and frequently geographically distant from each other. This will be accomplished while honoring flexible, rigorous standards of remote performance and availability service levels while ensuring that local control of resources are not compromised.

A number of projects are being developed in the world, which implement Grid environment. Two examples of these are European Union's DataGrid and UK's MammoGrid. The DataGrid project will bring together researchers from Biological Science, Earth Observation, and High-Energy Physics where large-scale, data-intensive computing is essential. The needs of these fields over coming years will provide the data - the DataGrid project will provide the computational means to handle them. By developing the necessary software - middleware in Grid parlance – in collaboration with some of the leading centers of Grid technology around the world, the project will benefit from valuable know-how and experience. It will complement and help to coordinate on-going national Grid projects in Europe. In doing so, it will extend the state of the art in large-scale, data-intensive computing, and lay down a solid foundation for European industry to build on. CERN is one of the leading organizations in the world where an active research over Grid is being carried out.

1.3. European Center for Nuclear Research (CERN)

What is CERN? CERN is the world's largest particle physics center. Founded in 1954, the laboratory was one of Europe's first joint ventures, and has become a shining example of international collaboration. From the original 12 signatories of the CERN convention, membership has grown to the present 20 Member States. CERN explores what matter is made of, and what forces hold it together. The Laboratory provides state-of-the-art scientific facilities for researchers to use. These are accelerators, which accelerate tiny particles to a fraction under the speed of light, and detectors to make the particles visible.

What difference does it make to us? Ever since the dawn of civilization, people have endeavored to learn more about their Universe. The goal is simply to learn but practical benefits often come later. In the 19th Century, Michael Faraday was asked by a

skeptical member of the British government what was the use of his work on electricity. His reply showed great foresight: "One day, Sir," he said, "You may tax it."

Just as Faraday was driven by the desire to know, the quest for pure knowledge at CERN drives technology forward. CERN has given the world advances as varied as medical imaging and the World-Wide Web. But the scientists responsible for these developments were not interested in medicine or computers. Their motivation was simply to find out.

CERN also plays an important role in advanced technical education. A comprehensive range of training schemes and fellowships attracts many talented young scientists and engineers to the Laboratory. Most go on to find careers in industry, where their experience of working in a high-tech multi-national environment is highly valued.

What are the experiments like? The experiments are like no others in the history of science. Designed and operated by hundreds of scientists, these experiments are often as big as houses. They run around the clock for several months a year, frequently taking years to complete.

Who works there? CERN employs just under 3000 people, encompassing a wide range of skills and trades - engineers, technicians, craftsmen, administrators, secretaries, workmen, ... The CERN staff designs and builds CERN's intricate machinery and ensures its smooth operation. It helps prepare, run, analyze and interpret the complex scientific experiments and carries out the variety of tasks required to make such a special organization successful. Some 6500 scientists, half of the world's particle physicists, come to CERN for their research. They represent 500 universities and over 80 nationalities.

Where is CERN? CERN is on the border between France and Switzerland, just outside Geneva. Its location symbolizes the international spirit of collaboration, which is the reason for the laboratory's success.

Did you know?

- That antimatter is routinely produced at CERN (more than 10 million particles per second).
- That the world's largest magnet, weighing more than the Eiffel tower, is at CERN.
- That the vacuum in the Laboratory's accelerators is the best between CERN and the moon.

- That CERN's biggest accelerator is 27 kilometers around, and particles travelling near the speed of light lap it over 11000 times each second.
- That CERN's detectors are the size of four-storey houses.
- That over 1800 physicists work on the biggest experiment being prepared for CERN's next accelerator. This experiment will generate data at a rate about equal to everyone on Earth simultaneously making 10 telephone calls each.

CERN's Involvement in Grid. The DataGrid initiative is led by CERN, together with five other main partners and fifteen associated partners. The DataGrid project brings together the following European leading research agencies: the European Space Agency (ESA), France's Centre National de la Recherche Scientifique (CNRS), Italy's Istituto Nazionale di Fisica Nucleare (INFN), the Dutch National Institute for Nuclear Physics and High Energy Physics (NIKHEF) and UK's Particle Physics and Astronomy Research Council (PPARC). The fifteen associated partners come from the Czech Republic, Finland, France, Germany, Hungary, Italy, the Netherlands, Spain, Sweden and the United Kingdom. NUST is also a non-member associate with CERN in the DataGrid Project. NUST's collaboration with CERN is an excellent example of shared goal to study the sensor outputs of the Compact Muon Solenoid (CMS) sensor array. CERN's research interest lies with the high-energy physics and NUST's lies with the computational challenges related to manipulation, storage, retrieval and usage of large data sets. The current project is also a contribution to this.

1.4. File/Data Replication

Data replication is a key issue in Data Grid and can be managed in different ways and at different levels of granularity: for example, at the file level or object level. Data replication is an optimization technique well known in the distributed systems and database communities as a means of achieving better access times to data (data locality) and/or fault tolerance (data availability). This technique appears clearly applicable to data distribution problems in large-scale scientific collaborations, due to their globally distributed user communities and distributed data sites. As an example of such an environment, we consider the High Energy Physics community where several thousand physicists want to access the Terabytes and even Petabytes transferring

Objectivity database files. File Replication in Data Grid is still under research, although certain prototypes like GDMP do exist.

Besides these replication issues routing the resources only to the desired users is also important to achieve bandwidth efficiency. To save user from enervate searches the lookup and retrieval of the resource can be made autonomous by using Software Agents. The bandwidth utilization can further be reduced, if instead of the resource itself, a metadata describing the resource is sent. These facts can be exploited to develop an Agent based application, which implements an efficient file transfer.

1.5. Concept of the Project

In the global networked world, dedicated servers exist at all local area networks. These local servers are responsible for servicing a particular subset of users and also for sharing the local resources to local users, with a possibility of connecting to outside world through standard protocols like www, ftp etc. Every user has to actively take part in searching for the resource they are interested in and transferring the resource to their own machine (or shared storage media). This procedure may require authentication of the user by the resource-controlling machine, something that is frequently not accessible outside an organization. There currently exists no mechanism that can autonomously locate these resources and route them to the user across the hard boundaries of organizations.

It is also possible that two different users on the same network segment retrieve the resource themselves thus wasting the bandwidth and storage resources at local server.

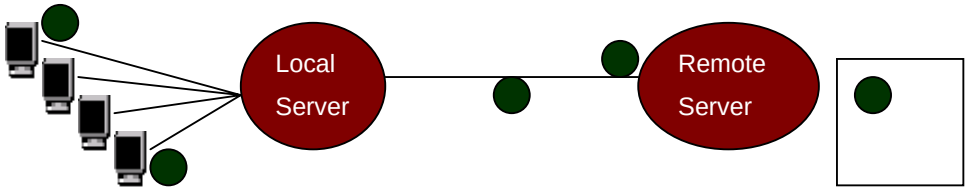


Figure 1 – Multiple Resource Replication

This project is designed to operate in CERN's data grid environment. The huge computational resources at CERN will produce some results, which may be required by

other stations at any tier. The flow of these results will be mostly down wards i.e. from CERN to regional centers, from regional centers to farm centers and from farm centers to clusters/nodes. In addition the flow of information can also be lateral as well as upwards. But whatever the case may be; we suppose that every station knows where to forward the information next. This project is basically automation of flow of these results i.e. the user just tells its agent to get a particular result whenever it is generated. Now to contact other agents and/or to stay on look out for the asked result is the agent's job. Therefore for the user the gathering of results is automated.

The mechanism implemented in this project will also help to reduce the network traffic. This is because here the complete results are not being passed to every subscriber rather only a template (metadata) is passed to subscribers. Basing on template the agent decides if the information is of use to it or otherwise. Since the templates do not contain the actual results, rather only their metadata, therefore the size of the template will not be very large (in bytes). However, the number of subscribers may be hundreds or even thousands.

1.6. Benefits of the System

The key problem that this system tackles is that of information access and particularly that of information discovery, information filtering and knowledge retrieval. What is truly novel about this is that it allows for collaboration across organizational boundaries, in a pseudo peer-to-peer manner without the need for bilateral security arrangements.

The underlying implementation is efficient in terms of bandwidth utilization as:

- Only meta-information is sent to the registered parties.
- Resource replication is carried out using a heuristic that can query nearby resources and the local regional center first. Thus the resources are requested from local resources as opposed to remote resources wherever possible.

Chapter

2

2. Project Specifications

2.1. Aim of the Project

To ensure automated resource announcement and replication in a distributed Grid like environment, with mechanisms for security and access control.

2.2. Objectives of the Project

- To carryout study of Grid, more specifically the Grid project at CERN.
- To study various technologies like Software Agents, Jini, LDAP, JESS etc to explore their features which could assist in achieving the aim.
- To design the application in such a manner that it contributes positively to CERN's CMS (Compact Muon Solenoid) project.
- To implement the project as per the specifications within the prescribed time.

2.3. Scope

The project was implemented in two increments. In the first phase, a prototype of the envisaged system was developed. In the second increment the remaining functionality was added to make it a complete running application. The project includes the following components:

- Storage of Template (the metadata of resource) into a directory.
- Notification of the creation of these resources to remote platforms.
- The matching of incoming templates with the knowledge base at the remote platform to ascertain if the information is needed.

- Transfer of resources.

The scope of this project does not cover the following issues. These will be addressed in follow-on projects:

- Parsing of the resource to automatically generate its template (metadata).
- Coordination of resource replication with other platforms.
- The extension of this mechanism to non-PC devices (such as cell phones, PDA etc)
- Implementation of complex template-profile matching algorithms in order to maximize relevance and focus recall.
- Integration with Globus (the Grid toolkit).
- Complex security and authentication of subscribers.

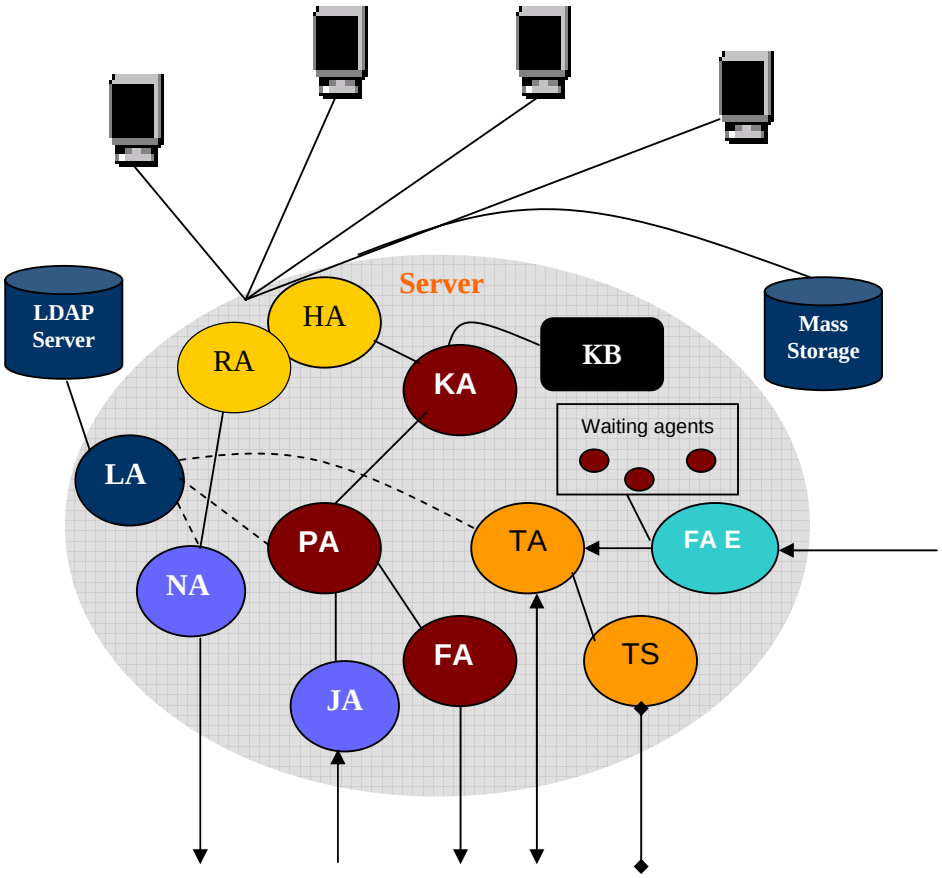
2.4. Project Outline

In a Grid like environment there are certain nodes, which need a resource while there are others, which possess or generate these resources. A resource can be a new data from sensors, a research document, an analysis data, an XML file etc. What qualifies any file or database as a resource to be shared will depend on local policies and rules. Whenever a new resource is generated its template (metadata about the resource) is made and passed along with the notification to registered clients.

The Jini Agent at the remote node receives this resource notification (template and resource mapping information). The Jini Agent then requests its Platform Agent (PA) to check the local knowledge base to ascertain whether any local user's interest profile matches the resource metadata. PA asks the local Knowledge Agent to perform this task. In the event that the resource matches a user interest profile, the PA generates a Fetch Agent (FA), giving it the complete requirement statement along with necessary authentication information, and instructs it to go and get the resource.

The Platform Agent on the remote node authenticates and authorizes the incoming Fetch Agent. A standard Security Authentication service may be used to do this task for the Platform Agent. The PA checks the mass storage for the availability of the resource requested by the Fetch Agent. The PA then hands over the Fetch Agent, along with resource mapping information, to the local Transaction Agent (TA). The TA activates its Transaction Service (TS) and asks the remote TA to start its TS. TS then initiates the

data transfer and on completion informs the Fetch Agent to go back. This has been indicated in the diagram given below:



Key

KA	Knowledge Agent	**KB**	Knowledge Base
RA	Resource Agent	**PA**	Platform Agent
HA	Home Agent	**NA**	Notification Agent
JA	Jini Agent	**LA**	LDAP Agent
FA E	Fetch Agent (External Platform)	**FA**	Fetch Agent
TA	Transaction Agent	**TS**	Transaction Service

Figure 2 – Interaction between various components of KMS

2.5. Component Description

This section profiles the functions of the required Grid Services and Intelligent Agents. The description of various components of the application are given below:

2.5.1. Resource Agent (RA)

- On generation of a resource the user specifies resource meta information on a GUI provided by the RA.
- The RA passes this information to Notification Agent at the local server.
- The RA will be activated whenever a user on a node starts this application and clicks to specify the attributes of a resource generated.

2.5.2. Notification Agent (NA)

- The NA receives the resource metadata from RA, fills in the remaining information and generates a template.
- It saves the template in the LDAP Server through the LDAP Agent.
- It then generates a notification along with the template to all registered Jini Agents.
- It uses Jini service for these notifications.
- It remains active on the server from the start of the application.

2.5.3. Home Agent (HA)

- HA provides a GUI to the user to specify his required information attributes.
- The HA then communicates with the Knowledge Agent so that the specified user need is stored in the Knowledge Base of the Knowledge Agent.
- The HA will be activated whenever a user on a node starts this application and clicks to specify a requirement.

2.5.4. Jini Agent (JA)

- The Jini Agent runs a Jini Service to receive Jini notifications.
- On receipt of a notification with a template from the NA it passes the template to the Platform Agent.
- It always remains active on the server from the start of the application.

2.5.5. Platform Agent (PA)

- The Platform Agent is a resident agent, which is live and active from the start of this application on the server.
- It receives templates from Jini Agent and passes them on to LDAP Agent for storage in the LDAP Server.
- It also instructs Knowledge Agent to compare the received template against the local knowledge base to ascertain whether the generated resource is required.
- If a resource is needed, it tasks a Fetch Agent (giving it necessary resource and authentication information) to retrieve the resource.
- It receives and authenticates Fetch Agents from remote networks and confirms the availability of the required resource.

2.5.6. Knowledge Agent (KA)

- KA is also a continuously running agent at the server.
- It receives information request from the HA and stores them in the form of rules in its knowledge base.
- Whenever a template is passed to Knowledge Agent it compares the template with its Knowledge Base.
- If the template reveals information of interest it passes the asking user ID along with template ID to the Platform Agent.

2.5.7. Transaction Agent (TA)

- TA is a continuously active agent at the server.
- On receiving a request from the incoming Fetch Agent it opens a server socket and communicates with the remote TA to start its socket.
- It then generates a Transaction Service for the transfer of resource.

2.5.8. Transaction Service

- Transaction Service is responsible for initiating the data transfer and publishing its status information.
- After the data transfer completes successfully it hands over the Completion Flag along with data check sum to Fetch Agent and asks it to go back.
- The Transaction Service also keeps a record of all transactions in process and in case of failures resumes the file transfer.

2.5.9. Fetch Agent (FA)

- The Platform Agent, in order to bring a resource from remote node, activates the Fetch Agent. It receives the Template ID, authentication and transaction information as part of the initialization.
- The Fetch Agent travels to the Platform Agent and passes on the request. It then helps in starting the data transfer and waits for the completion signal.
- The Fetch Agent returns to its home node, bringing back the CRC (which will be used to verify the data).

2.5.10. LDAP Agent (LA)

- LA is also a continuously active agent at the server.
- It is basically a link between the LDAP Server and the other agents, which need to retrieve or store some information from the LDAP Server.

2.6. Work Plan

This section describes the plan for the design and implementation of the project. The project is being implemented following the incremental model and the work packages. In all there are two increments and eight work packages. Each work package has well defined objectives and contains a list of Deliverables and Milestones. The first increment covers 60% of work and is to be completed by 16 Mar. The second increment covers the remaining 40% work and is to be completed by 20 Apr 2002.

2.6.1. Time Lines

Activity	**Completion**
• Project to be implemented in two increments	
o First Increment	16 Mar
o Second Increment	20 Apr
• First Increment	
o Knowledge Acquisition	19 Jan
o Analysis and Design	9 Feb (3 weeks)
o Coding and Integration	2 Mar (3 weeks)
o Testing	16 Mar (2 weeks)
• Second Increment	
o Analysis and Knowledge Acquisition	20 Mar
o Design	23 Mar
o Coding and Integration	6 Apr (2 weeks)
o Testing	20 Apr (2 weeks)

2.6.2. Work Packages

The work packages have been listed below. First four work packages describe the work for Increment 1, while remaining four work packages describe the work for Increment 2. The time distribution to these work packages is shown in the form of a Gantt Chart (Figure 3).

WP1 – Concept Analysis and Knowledge Acquisition
WP2 – Increment 1 – Analysis and Design
WP3 – Increment 1 – Implementation and Integration
WP4 – Increment 1 – Testing
WP5 – Increment 2 – Analysis and Design
WP6 – Increment 2 – Implementation and Integration
WP7 – Increment 2 – Testing
WP8 – Industry Exploitation and Research Paper

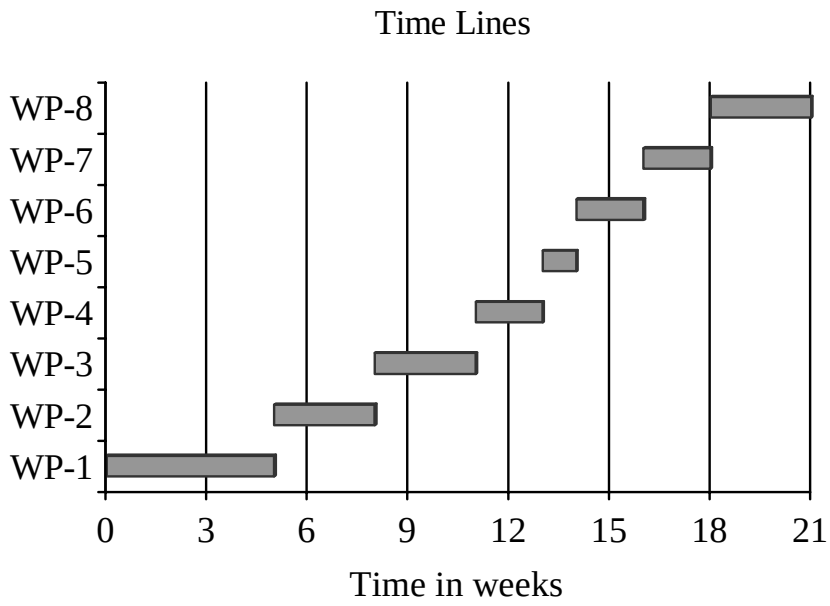


Figure 3 – Gantt Chart for Work Packages

The details of these work packages are given in the subsequent pages. The deliverables for each work package are attached as appendices.

WP1 – Concept Analysis and Knowledge Acquisition

Work package number: 1
Starting date: 17 Dec 2001
Finishing date: 19 Jan 2002

Objectives

1. Analyze the project concept and see the appropriateness of technologies selected.
2. To plan the developmental phases of the whole project.
3. To carryout in-depth study of the selected technologies.

Description of Work

T1.1 Defining the Project

- In this task a basic concept of the project will be evolved to a concrete project statement.
- Classify the components of the project according to the technologies (i.e. software agents, JINI, LDAP) in which they will be implemented.

T1.2 Project Management

- Review the project scope. Divide the project into two Increments and distribute the work into different Work Packages.
- Define the objectives and describe the work in each Work Package.
- Prepare a list of deliverables and milestones.
- Decide timelines for Work Packages.

T1.3 Knowledge Acquisition

- Explore the project and see the appropriateness of selected technologies.
- In-depth study of technologies according to the project statement.
- To experiment and carryout practice exercises in respective technologies.

Deliverables

D1.1 – Report on project concept.
D1.2 – Technical papers on selected technologies.

Milestones and expected result

M1.1 – Basic concept of the project will be developed.
M1.2 – The required expertise in respective technologies will be completed.

WP2 – Increment 1 – Analysis and Design

Work package number: 2
Starting date: 21 Jan 2002
Finishing date: 9 Feb 2002

Objectives

1. To define the boundaries of first increment.
2. To define data flow based on use case scenarios.
3. To design system architecture, which ensures that all functional and non-functional requirements and constraints identified in WP1 can be satisfied.

Description of Work

T1.1 – Requirement elicitation

- In this task the boundaries and scope of the first increment will be decided.

T2.2 – Development of the Use Case Model

- This will result in identifying and describing the actors.

T2.3 – Development of System Models

- This will include Flow Diagrams, Class Models and Sequence Diagrams. These diagrams will describe the static and dynamic behavior of the first running prototype.

Deliverables

D2.1 – Report on scope of First Increment.
D2.2 – Software Requirements Specification document covering following:

- Use Case Model
- Flow Diagrams
- Class Relationship Diagrams
- Sequence Diagrams

Milestones and expected result

M2.1 – Requirements analysis for increment-1 complete.
M2.2 – System Design for increment-1 available.

WP3 – Increment 1 – Implementation and Integration

Work package number: 3
Starting date: 11 Feb 2002
Finishing date: 2 Mar 2002

Objectives

1. To implement the specifications formulated in WP2.
2. Integrate various components of the system to check the functionality in the overall scenario of first increment.
3. To demonstrate a basic functional prototype, which would include the essential features to prove the concept.

Description of Work

T3.1 – Implement appropriate API

- Implementation of functionality defined for agents in increment 1.
- Simultaneous development of LDAP basic operations and resource notification and transfer by JINI.

T3.2 – Integration of software agents, LDAP and JINI services.

Deliverables

D3.1 – A running prototype showing the essential features of the project.
D3.2 – Report on the code covering the major procedures used to implement the functionality.

Milestones and expected result

M3.1 – Implementation phase of Increment 1 completed and a working prototype is available for testing.

WP4 – Increment 1 – Testing

Work package number: 4
Starting date: 4 Mar 2002
Finishing date: 16 Mar 2002

Objectives

1. Carryout detailed white box testing of prototype 1.
2. Carryout detailed black box testing of prototype 1.
3. Amend the code to rectify the shortcomings observed.

Description of Work

T4.1 – White Box Testing

- Ensure that all independent paths within a module have been exercised at least once.
- Exercise all logical decisions on their true and false sides.
- Execute all loops at their boundaries
- Exercise internal data structures to assure their validity.
- Amend the code as needed.

T4.2 – Black Box Testing

- Focus on the functional requirements of the software.
- To ascertain what data rates and data volume can the system tolerate?
- To ascertain what effect will specific combinations of input data have on system operation?
- Amend the code as needed.

Deliverables

D4.1 – A fully tested and running prototype.
D4.2 – Report covering the salient observations from testing

Milestones and expected result

M4.1 – Increment 1 completed
M4.2 - A fully tested and running prototype is available for further evaluation in increment 2.

WP5 – Increment 2 – Analysis and Design

Work package number: 5
Starting date: 18 Mar 2002
Finishing date: 23 Mar 2002

Objectives

1. To analyze the outcome of increment 1 and set goals for the increment 2, (the last increment).
2. To enhance system design, which overcomes the shortcomings and fulfills the leftover requirements.

Description of Work

T5.1 – Requirement Analysis

The extensions and scope of the second increment will be decided.

T5.2 – Design

The design of the classes needed for the second increment will be made. This will include Class Models and Sequence Diagrams to describe the overall behavior of the final deliverable

Deliverables

D5.1 – Report on scope of Second Increment.
D5.2 – Software Requirements Specification document covering following:

- Class Modeling
- Sequence Diagrams

Milestones and expected result

M5.1 – Requirements analysis for increment-1 as well as for the entire project is complete.
M5.2 - Design for increment-2 is complete

WP6 – Increment 2 – Implementation and Integration

Work package number: 6
Starting date: 25 Mar 2002
Finishing date: 6 Apr 2002

Objectives

1. To implement the specifications formulated in WP5.
2. Integrate all the components of the system to check the overall functionality.
3. To complete the implementation as conceived in WP1.

Description of Work

T6.1 – Implementation of Increment-2

Implementation of the remaining API of Software Agents, JINI and LDAP. This includes the coding of new classes and any changes in the implementation of existing classes.

T6.2 – Integration of new components with those already developed in Increment-1.

Deliverables

D6.1 – A fully running prototype meeting the complete functionality of the project.

D6.2 – Report on implementation of second increment.

Milestones and expected result

M6.1 – Complete project is implemented and ready for final testing

WP7 – Increment 2 – Testing

Work package number: 7
Starting date: 8 Apr 2002
Finishing date: 20 Apr 2002

Objectives

1. To carryout detailed testing (including white box testing and black box testing) of the complete project.
2. To rectify the code if any faults or bugs have occurred.

Description of Work

T7.1 – White Box Testing

Testing of only those components that are implemented in the second increment.

T7.2 – Black Box Testing

Testing the entire application to ensure that all inputs are processed autonomously and the desired output is achieved.

T7.3 –User Testing

Testing by a third party (not related with the development) to get their opinion about the utility and friendliness of the application.

Deliverables

D7.1 – A fully tested and running application.
D7.2 – Report on the results of the Testing of increment-2 and the complete project.
D7.3 – A final package including the deliverables (all written reports and the application software) of the project.

Milestones and expected result

M7.1 – Verification of final system functionality and compliance with specifications.
M7.2 – Project Completed !!!

WP8 – Industry Exploitation and Research Paper

Work package number: 8
Starting date: 22 Apr 2002
Finishing date: 10 May 2002

Objectives

1. To publish an international research paper.
2. To exploit the industry to see its feasibility and possible application in various domains.

Description of Work

T8.1 – Writing a Research Paper

Since it is a research project, a research paper will be written at the end of the project, to evaluate this application and include all the inferences drawn.

T8.2 – Industry Exploitation

The application will be handed over to various professional bodies for their evaluation and comments on its usability.

Deliverables

D8.1 – A research paper.
D8.2 – Report on the feedback from the industry to determine the future of this research.

Milestones and expected result

M8.1 – Work projected on international forum and new avenues opened up for further research in Agent and Grid Technologies

Chapter 3

3. Software Agents

3.1. What are Software Agents

Software Agents are considered as objects that are able to move autonomously in a network of hosts to fulfill their tasks. Agents are able to decide, based on their local knowledge, if or when or where to migrate in the network. Therefore an agent has following basic characteristics:

- Intelligence
- Mobility
- Ability to communicate to other agents

An agent is capable of acting intelligently on behalf of a user or users in order to accomplish a task.

3.2. History of Software Agents

The history of agents can be traced back along with the history of spying. But the history of software agents is not so deep. The invent of networks lead to the need of efficient network software. Remote Procedure Call (RPC) or Remote Method Invocation (RMI) enabled access to piece of code at remote platforms. But to get a job completed at a remote station it may involve a number of calls and therefore increased network traffic. The agents were derived from the fact that instead of calling the remote methods why not to send an object, which goes to remote station with a well defined task. This object executes itself at remote platform and using functionality implemented at remote station completes its task.

The evolution of agents can be divided into two strands: to the study that has been done before nineties and studies after that. In the seventies, Carl Hewitt proposed a concept of "…self contained, interactive and concurrently executing object…" that he called "actor". This object had an internal state and could communicate with other similar objects. This work was the base for studies done with multi agent systems that was mainly concerned with macro issues. The aim of the studies was to analyze and specify systems containing multiple collaborative agents. This approach gives emphasis to society of agents over individual agents. Later issues researched were theoretical issues like architectural and language problems. Although there is inevitably some overlap with the issues researched in the nineties, new type of research has clearly emerged. Previously only research was done on macro level, but now new research has been done on broader range of agent types (or classes). This can also be credited to the fact that more and larger companies have started getting interested in agents – also the term 'agent' is being used more and more broadly than before. Nowadays almost every piece of software has some 'intelligence' in it – be it mails filtering, adaptive interfaces or help with writing a letter – and especially if the intelligence can be seen as an entity, it's usually called an agent. It has been predicted, that in few years time most of the consumer products (not just software) will have some kind of embedded agents in them.

3.3. Why Mobile Agents

The following is taken from Seven Good Reasons for Mobile Agents.

- They reduce network load. Distributed systems often require many messages to achieve a task. With agents you send to where the work needs to be done. The agent then works locally.
- They overcome network latency. Since the agent is working locally on the task, it does not have to deal with network latency.
- They encapsulate protocols. Distributed systems use protocols to define how messages and data are transferred. To modify the protocol requires changing the code on all the machines in the system. With agents, the protocol is just accepting an agent and let it work. So creating a new agent can create new functionality.

- They execute asynchronously and autonomously. Mobile devices are often not continuously connected to a network. Systems that require open connections will not work on mobile devices. With agents, a mobile device can connect to the network to check for work/messages. An agent can be sent to the device and work even after the device disconnects. The agent can wait until the device is reconnected to report the result of its task.
- They adapt dynamically. Agents can distribute themselves around on machines on the network to best solve the task at hand.
- They are naturally heterogeneous.
- They are robust and fault-tolerant. If a host is being shut down, agent can move on to another host to continue to operate.

3.4. Agent Concepts

The first commercial implementation of the mobile agent concept, General Magic's Telescript technology, attempted to allow automated as well as interactive access to a network of computers using mobile agents. The commercial focus of General Magic technology, the electronic marketplace, requires a network that will let providers and consumers of goods and services find one another and transact business electronically. Although the electronic marketplace still does not exist fully, the Internet has already encouraged its beginnings. Telescript's creators envision the electronic marketplace as only a small piece of the agent world that will exist in coming years. There, agents will act on their user's behalf to research information for work, find the best hotel for vacation, provide up-to-the-minute scores of sporting events, or simply send and receive messages between friends.

Since General Magic's realization of the basic concept of agent architecture through Telescript is an easy example to follow, we'll use it to introduce how an agent can work successfully. Telescript implements the following principal systems associated with remote programming: places, agents, travel, meetings, connections, authorities, and permits.

- **Places.** Agent technology models a network of computers, however large, as a collection of places offering a service to the mobile agents that enter. Servers provide some places and user computers provide others.
- **Agents.** Communicating applications are modeled as a collection of agents.
- **Travel.** Travel allows an agent to obtain a service offered remotely and then return to its starting place.
- **Meetings.** A meeting lets agents in the same computer call one another's procedures.
- **Connections.** A connection, when two agents in different places communicate, is often made for the benefit of the human users of interactive applications.
- **Authorities.** This agent system lets one agent or place discern the authority of another. The authority of an agent or place in the electronic world is the individual or organization in the physical world that it represents.
- **Permits.** A permit is data that grants capabilities. An agent or place can discern its capabilities, what it is permitted to do, but cannot increase them.
- **Putting things together.** An agent's travel is not restricted to a single round-trip. The power of mobile agents becomes fully apparent when one considers that an agent may travel to several places in succession. Using the basic services of the places it visits, such an agent can provide a higher-level, composite service.

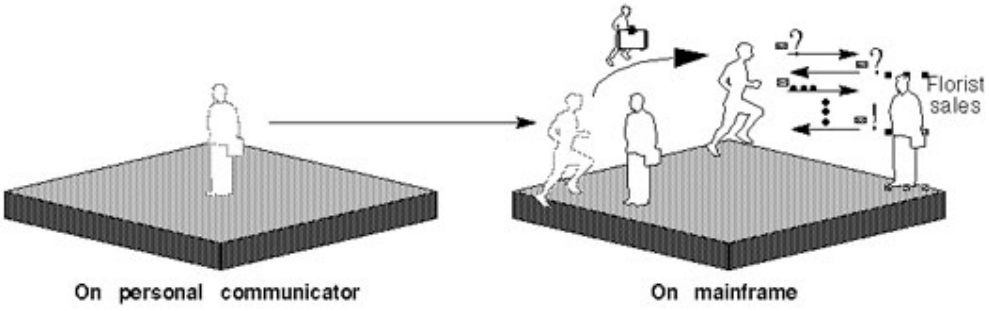


Figure 4 – Agent Interactions

3.5. Agent Frameworks

A number of Agent Frameworks to develop Agent based systems are available. Before starting to develop an agent based application it is very important to understand these frameworks in detail and select the one, which is most appropriate to own needs. Some of the frameworks are:

- IBM Aglets
- Mobile Agent Platform (MAP)
- Voyager
- Concordia
- Fipa-os

3.5.1. IBM Aglets

The Java aglet extends the model of network-mobile code made famous by Java applets. Like an applet, the class files for an aglet can migrate across a network. But unlike applets, when an aglet migrates it also carries its state. An applet is code that can move across a network from a server to a client. An aglet is a running Java program (code and state) that can move from one host to another on a network. In addition, because an aglet carries its state wherever it goes, it can travel sequentially to many destinations on a network, including eventually returning back to its original host.

A Java aglet is similar to an applet in that it runs as a thread (or multiple threads) inside the context of a host Java application. To run applets, a Web browser fires off a Java application to host any applets it may encounter as the user browses from page to page. That application installs a security manager to enforce restrictions on the activities of any untrusted applets. To download an applet's class files, the application creates class loaders that know how to request class files from an HTTP server.

Likewise, an aglet requires a host Java application, an "aglet host," to be running on a computer before it can visit that computer. When aglets travel across a network, they migrate from one aglet host to another. Each aglet host installs a security manager to enforce restrictions on the activities of untrusted aglets. Hosts upload aglets through class loaders that know how to retrieve the class files and state of an aglet from a remote aglet host.

3.5.2. Mobile Agents Platform (MAP)

The MAP is a mobile agent platform, which has been created in order to provide all the basic tools for the creation, the management, the migration of agents, and the communication among them. In fact, it enables us to create, run, suspend, wake up, deactivate, reactivate agents, to stop their execution, to make them communicate with each other and migrate through the network. Furthermore, the MAP enables the remote creation of agents that can be dynamically loaded and run even on hosts where the corresponding class is not present. The MAP is also equipped with a user-friendly graphic interface that facilitates the access to the above mentioned management functions.

The MAP platform complies with MASIF; the functions it implements therefore comply with the set of interfaces and functionalities defined in the standard. This way, each MAP platform is able to accept agents coming from other platforms (that also comply with MASIF) and make them run, thus enabling them to access the methods needed for their management. The same way, a MAP agent is allowed to migrate towards other platforms able to support it, and is also allowed to run on them.

3.5.3. Voyager

Voyager is framework developed by ObjectSpace inc. It provides following features:

- Remote-Enabling a Class. No need for a class to implement java.rmi.Remote. All classes can be used remotely.
- Client-side Startup. Client can create a server object remotely.
- Dynamic Class Loading. Voyager orb (or daemon) has a build-in http server.
- Distributed Garbage Collection
- Dynamic Aggregation. This feature allows you to add secondary objects (termed facets) to a primary object at runtime. For example, you can dynamically add hobbies to an employee, a repair history to a car, or a payment record to a customer.
- CORBA, RMI, DCOM. There is full native support for IDL, IIOP, and bidirectional IDL<->Java translation. No stub generators or helper classes are required. Full support for RMI. Soon it will support DCOM.

- Mobility. You can move any serializable object between programs at runtime. 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 resent.
- Autonomous Mobile Agents.
- Activation. The activation framework allows objects to be persisted to any kind of database and automatically re-activated in the case that the program is restarted. An object does not have to be modified in any way to be activatable.
- Applets and Servlets. Voyager-enabled applets and servlets are supported
- Naming Service. Voyager orb (or daemon) has a build-in naming service
- Multicast to Distributed Java Objects. You can multicast a Java message to a distributed group of objects without requiring the sender or receiver to be modified in any way.
- Publish-Subscribe of Remote Events. You can publish a Java event on a specified topic to a distributed group of subscribers. The publish-subscribe facility supports server-side filtering and wildcard matching of topics.
- Advanced Messaging. You can send oneway, sync, and future messages. Oneway messages return immediately and discard the return value. Future messages immediately return a placeholder to the result, which may then be polled or read in a blocking fashion.

3.5.4. Concordia

The Concordia Java-based mobile agent systems framework was developed to address the needs of the mobile user. Concordia is the most comprehensive product among commercial offerings and can be tailored to fit the particular hardware needs of the mobile user due to its modular means of deployment, i.e., only the required components need to be installed. Concordia can be deployed on a spectrum of hardware devices, from smartphones and PDAs (Personal Digital Assistants) to high-end back-room servers to fulfill the needs of mobile users for enterprise wide computing.

Background & Objective: The Concordia Java-based mobile agent systems framework was developed to address most of the needs that earlier systems such as General Magic's Telescript could not provide for the mobile user. The goal of this

project was to provide complete mobile agent systems support for the mobile user within the context of enterprise-wide computing.

Technical Discussion: Concordia is the most complete mobile agent systems framework available among all commercial products and research prototypes in the Java-based mobile agents space. Concordia offers complete systems reliability for agent communication, execution, and transmission, and server robustness in the form of seamless restart and recovery upon system failure. Furthermore, Concordia offers the most complete security among all mobile agent systems offerings. Its security support provides protection of the agent from tampering by other agents, protection of access to server system resources by unauthorized agents, and protection of agents during transmission via encryption techniques. Concordia is highly scalable and can be deployed across a spectrum of hardware devices, from smartphones and PDAs to high-end backroom servers; its memory requirements can be tailored to a particular device and solution by deploying only those components that are needed by a particular customer. Concordia currently comes in two distinct flavors: the Full Server version and a Lightweight Server version (targeted for embedded devices and requiring under 3.0MBs of memory for the Concordia Server).

3.5.5. Fipa-os

This is the framework, which has been actually used to develop the agent framework of the project. It has been explained in detail in the next section.

3.6. Fipa-os Architecture

3.6.1. Introduction

FIPA-OS (FIPA Open Source) is an open agent platform originating from Nortel Networks. The platform supports communication between multiple agents using an agent communication language, which conforms to the FIPA (Foundation for Intelligent Physical Agents) agent standards. A key focus of the platform is that it supports openness. This is naturally supported by the agent paradigm itself and by the design of the platform, whose parts have loose coupling such that extensions and innovations to

support agent communication can occur in several key areas. The openness is further emphasized in that the platform software is distributed and managed under an open-source licensing scheme. FIPA-OS is being deployed in several domains including virtual private network provisioning, distributed meeting scheduling and a virtual home environment. It has been demonstrated to interoperate with other heterogeneous FIPA compliant platforms and is in use in numerous institutions around the world.

In the context of FIPA, an agent is an encapsulated software entity with its own state, behavior, thread of control, and an ability to interact and communicate with other entities – including people, other agents, and legacy systems. This definition puts an agent in the same family as objects, functions, processes, and daemons but it is also distinct in that it is at a much higher-level of abstraction. The agent interaction paradigm differs from the traditional client-server approach: agents can interact on a peer-to-peer level, mediating, collaborating, and co-operating to achieve their goals.

A common (but by no means necessary) attribute of an agent is an ability to migrate seamlessly from one platform to another whilst retaining state information, a mobile agent. One use of mobility is in the deployment and upgrade of an agent. Another common type of agent is the intelligent agent, one that exhibits 'smart' behavior. Such 'smarts' can range from the primitive behavior achieved through following user-defined scripts, to the adaptive behavior of neural networks or other heuristic techniques. In general, intelligent agents are not mobile since; in general, the larger an agent is the less desirable it is to move it; coding artificial intelligence into an agent will undoubtedly make it bigger. There is an exception to this last statement, 'Swarm' intelligence. This is a form of distributed artificial intelligence modeled on ant-like collective intelligence. The ant-like 'agents' collaborate to perform complex tasks, which individually they are unable to solve due to their limited intelligence (e.g. ant-based routing) (Schoonderwoerd et al,1996). Another prevalent, but optional, attribute of an agent is anthropomorphism or the 'human factor': this can take the form of physical appearance, or human attributes such as goal-directed behavior, trust, beliefs, desires and even emotions. There are three important agent standardization efforts which are attempting to support interoperability between agents on different types of agent platform: KQML community, OMG’s MASIF and FIPA.

3.6.2. High Level Architecture

FIPA-OS is a component-orientated toolkit for constructing FIPA compliant Agents using mandatory components (i.e. components required by ALL FIPA-OS Agents to execute), components with switch able implementations, and optional components (i.e. components that a FIPA-OS Agent can optionally use). Figure 9 highlights the available components and there relationship with each other (NOTE: The Planner Scheduler is not currently available).

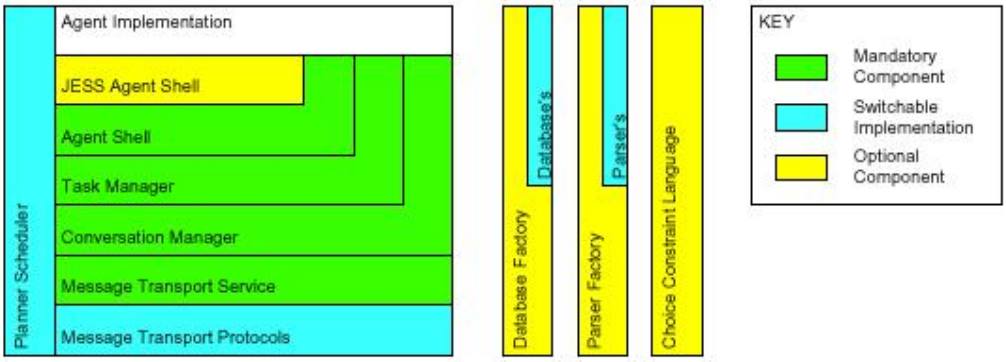


Figure 5 - Components within FIPA-OS

The Database Factory, Parser Factory and CCL components are optional and do not have an explicit relationship with the other components within the tool-kit. The Planner Scheduler generally has the ability to interact with all components of an Agent, although not necessarily vice versa. The switchable implementations included as part of the FIPA-OS distribution for each component include:

- MTP's
  - RMI (proprietary)
  - IIOP (FIPA compliant)
- Database's
  - Memory Database
  - Serialization Database
- Parser's
  - SL
  - ACL
  - XML
  - RDF

Multi-Agent Systems (MAS) consists of many agents that can combine their abilities to solve problems. Due to the collaborative nature of MAS, agent standards play an important role for commercialization of agent technology. FIPA (Foundation for Intelligent Physical Agents), a non-profit organization for producing standards for open agent interfaces, has produced several specifications tackling different aspects of MAS. These specifications don't try to dictate internal architectures of agents or how they should be implemented, but they specify the interfaces necessary to support interoperability between different MAS. FIPA identifies four areas for standardization:

3.6.3. Agent Communication Interface

This describes the communication between agents and it supports all interactions between two agents. FIPA has specified an Agent Communication Language (ACL) to support the interface (ACL is based on Knowledge Querying and Communication Language KQML). ACL has five levels of formal semantics:

- Protocol – defines the structure of the agent dialogue, like fipa-request-protocol.
- Communicative Act (CA) – defines the type of communication currently performed, like "request" when an agent is requesting a service from another agent.
- Messaging – defines meta-information of the message, like identity of the sender and receiver.
- Content Language – defines the language (i.e. the grammar) of the content message, like XML.
- Ontology – defines the meaning of terms and concepts used in content expression like meaning of the XML tags.

3.6.4. Agent Management

This describes facilities necessary to support the creation of agents, communication between agents, as well as security and mobility. FIPA 97 defines the platform to be an infrastructure in which agents can be deployed and where FIPA agents can enter, advertise their services, locate other agents and communicate with agents of other

platforms. It consists of three agents – often called the platform agents – ACC, AMS and default DF:

- Directory Facilitator (DF) – DF is an agent that provides “yellow pages” services to other agents, where agents can register their services and request information of other agents.
- Agent Management System (AMS) – AMS is an agent that provides an agent name service, an index of all the agents currently registered in the platform. AMS makes sure that all the agents have unique names, Agent Global Identifiers (GUIDs). AMS has supervisory power on the platform and it can create, delete and de-register agents and it oversees the migration of the agents to and from other platforms.
- Agent Communication Channel (ACC) – ACC is an agent that routes messages between agent platforms and it must minimally support Internet Inter-Orb Protocol (IIOP).

3.6.5. Agent Lifecycle

FIPA agent lifecycle defines that the agent can be in three different states in its lifetime (represented in the AMS): active, waiting and suspended. Mobility support specification defines one more state to support the mobility (transit), and two actions to enter and leave the state (move and execute).

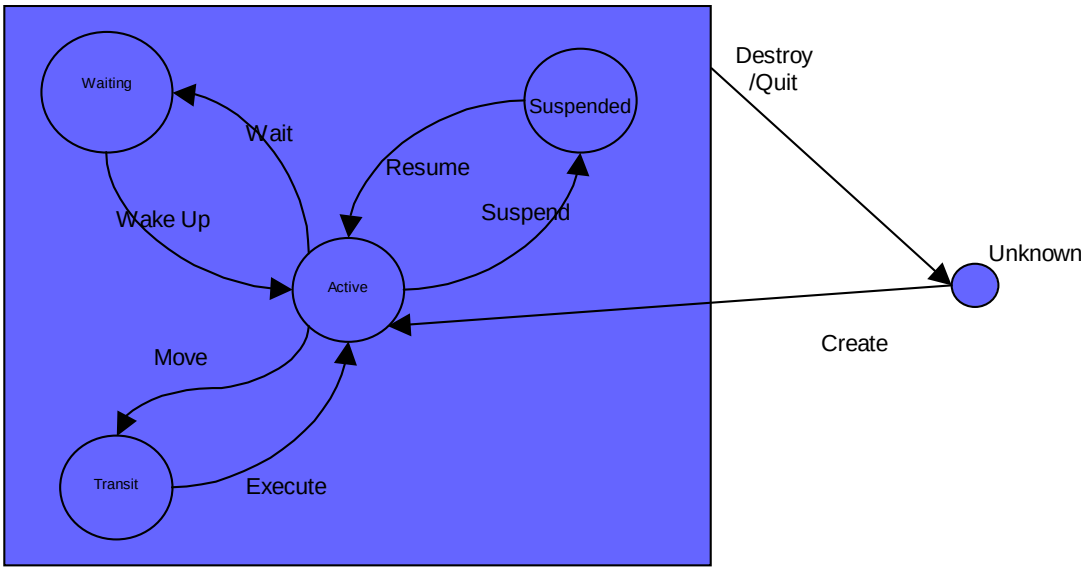


Figure 6 - Possible Agent States

3.6.6. Core Components

3.6.6.1. Non-Component Core Classes

The classes that any non-trivial Agent implementation will make use of, but are not necessarily part of any particular component are mentioned below:

- Fipaos.ont.fipa.ACL
- Fipaos.ont.fipa.fipaman.Envelope
- Fipaos.mts.Message
- Fipaos.util.DIAGNOSTICS

3.6.6.2. Agent Shell (FIPAOSAgent)

The FIPAOSAgent class provides a shell for Agent implementation to use by simply extending this class. The FIPAOSAgent shell is responsible for loading an Agent's profile, and initializing the other components of which the Agent is composed. It creates these mandatory components in this order initially:

- MTS
- Task Manager
- Conversation Manager

At initialization of the Conversation Manager, references to the MTS and Task Manager are passed to enable them to be dynamically bound to the CM. This is all achieved via the listener interfaces implemented by the various components, so these components are not explicitly dependant on each other.

The Agent Shell provides the following functionality:

- Sending messages – This is accomplished by using the forward() method in either the FIPAOSAgent or Task class, depending on where in an Agent implementation the message is being sent from. In the former case, the outgoing message is always passed to the CM via its sendMessage() method. See the Task Manager and Conversation Manager sections for details on how messages are dealt with.

- Retrieving the Agents' properties (Profiles, AID, state) & Locating platform Agents (DF and AMS) – numerous methods are provided to access this information from the FIPAOSAgent class.
- Registration with platform Agents – The FIPAOSAgent class provides registerWithAMS() and registerWithDF() methods, as well as the call-back methods registrationSucceeded(), registrationFailed() and registrationRefused()which should be overridden. This functionality is provided by use of the AMSRegistrationTask and DFRegistrationTask's 1 . Figure 14 highlights how the Agent Shell creates a AMSRegistrationTask to register with the AMS, and a callback is made to indicate the result of that registration (NOTE: this is only a logical representation of interactions, and doesn't reflect the concrete interactions that occur). Reception of incoming messages from the AMS by the Task Manager is implicit. A similar set of interactions occur when registering with the DF.

3.6.6.3. TM (Task Manager)

The Task Manager provides the ability to split the functionality of an Agent into smaller, disjoint units of works known as Tasks. The aim is that Task's are self-contained pieces of code that carry out some task and (optionally) return a result, have the ability to send and receive messages, and have little or preferably no dependence on the Agent they are executed within. This provides a number of benefits: These classes are not part of the FIPA-OSv1.3.2 distribution, but are available separately from our SourceForge CVS repository.

- Tasks are highly re-usable - they can be used in many Agents without having to re-write the same code / functionality.
- Easy to debug, since tracking the flow of control is simple (Task's are completely event-based) and useful debugging messages help to indicate when task-interactions fail/are unhandled.
- An Agent can execute multiple Tasks at once – the Task Manager takes care of routing incoming messages and other events to the right Tasks, rather than using a "cludge" of code within the Agent itself to decide what to do with a particular message.

- Conversation state is effectively encapsulated within a Task, reducing the manual tracking of Conversations to a bare minimum.
- Tasks can spawn child-tasks – this enables complex Task's to be created through simply utilising simpler Task within them.

3.6.6.4. CM (Conversation Manager)

The CM provides the ability to track conversation state at the performative level, as well as mechanisms for grouping messages of the same conversation together. If a conversation is specified as following a particular protocol, the CM will ensure that the protocol is being followed, by both the Agent it is part of, and the other Agent involved in the conversation.

Conversation objects represent individual conversations, and encapsulate all of the state information and messages sent and received as part of that conversation. Hence they perform the necessary validation of the protocol being used by the conversation, and provide mechanisms for discovering what messages have been sent/received, and the messages that should be sent next. The ConversationManager also has a reference to a Database implementation to enable Conversation objects to be stored once they are no longer active (i.e. when the conversation they represent has completed). A Map of active Conversation's is kept by the ConversationManager, enabling quick look-up upon receipt of a message. Various specializations of the Conversation class are provided to enable different protocols to be supported. Each specialization simply defines the protocol (in terms of performatives) to be followed for a particular conversation of that protocol type.

Protocol Definition. The protocol a particular conversation type follows is defined by specifying a class variable (__protocol) containing a tree-like structure defining the protocol. This is achieved through specifying an Object[] for each node in the tree, with details of what performative is expected next from which Agent in the conversation, what the desired action is (inform the Agent, ignore etc…) and references to its' child-nodes. The protocol definition can contain loops (although these will need to be closed using a static initialiser), and handling of "not-understood" messages is implicit.

3.6.6.5. MTS (Message Transport Service)

The MTS provides the ability to send and receive messages to an Agent implementation. The MTS within FIPA-OS is logically split such that incoming and outgoing messages pass through a number of services within a "service stack" (see Figure 21). Each service is a stand-alone component that performs some transformation on outgoing messages, and the inverse transformation on incoming messages. This model is used for the following reasons:

- Ideally each service performs its own function on incoming and outgoing messages – this enables the functionality of the MTS to be split into distinct decoupled components that can be individually tested (e.g. routing of messages to the ACC could be once service, whereas buffering messages could be another). Due to the non-trivial required behavior of the MTS, it is logical to break the implementation of the requirements into individual components, which in conjunction meet the overall requirements of the MTS.
- Addition of functionality to the MTS simply requires a new service to be created.
- Extra services can be slotted into the stack at runtime, due to lack of compile-time bindings between services.

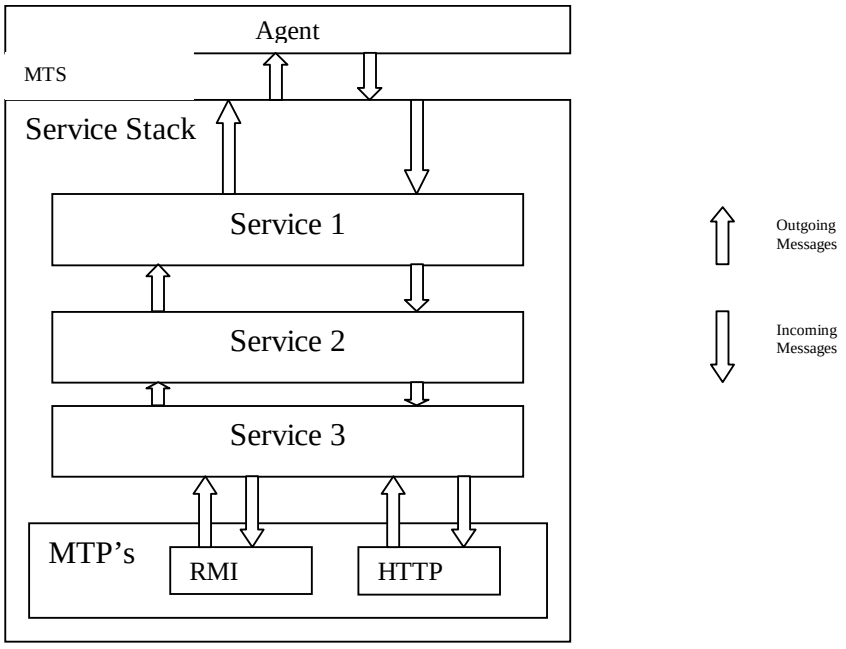


Figure 7 - Logical Composition of MTS

3.6.6.6. MTPs (Message Transport Protocols)

MTPs provide the mechanisms for sending and receiving messages from one Agent to another.

MTPBase Class. The MTPBase class contains functionality that is common across a number of MTPs. This includes handling incoming and outgoing messages, raising appropriate exceptions and error messages and other general behavior. The MTPBase class deals with {Envelope, Object} tuples, where theEnvelope determines the behavior of the MTP, and the Object is the payload of the message.

The InternalMTPBase and ExternalMTPBase classes specialize the MTPBase class to a particular type of MTP – either internal or external – and simply provides a translation mechanism between the InternalMTP and ExternalMTP interfaces and the functionality defined by the MTPBase class (i.e. providing the following translations respectively: Message _ {Envelope, Object} and {Envelope, byte[]} _ {Envelope, Object}).

Internal MTPs. An MTP generally falls into this category if:

- It provides a proprietary transport mechanism
- Aims to provide efficiency rather than inter-operability
- Does not require the message or its envelope to be prepared for its use (i.e. stringified or serialized in any form)

Internal MTPs are the main type of transport used by Agents within a platform, assuming that the majority of communications are intra-platform.

External MTPs. An MTP generally falls into this category if:

- It provides a standardised transport mechanism (i.e. following a particular FIPA specification)
- Aims to provide inter-operability rather than efficiency
- Requires the message is prepared in some form before it is passed to is (i.e. stringified or serialized in some form).

External MTPs are currently only used by the ACC (although this will change when MTS profiles are introduced, allowing individual Agents to make use of external transports).

Chapter 4

4. Overview of Jini

4.1. What is JINI Technology

Jini technology brings to the network the facilities of distributed computing, network-based services, seamless expansion, reliable smart devices, and ease of administration. Here's the vision: When you walk up to an interaction device that is part of a system employing Jini technology, all of its services are as available to you as if they were on your own computer--and services include not only software but hardware devices as well, including disk drives, DVD players, VCRs, printers, scanners, digital cameras, and almost anything else you could imagine that passes information in and out. Adding a new device to a system employing Jini technology is simply plugging it in.
In other words, the Jini technology infrastructure is a system architecture (hardware, software, and network) that supports the notion that a computing environment is a network-connected set of computing, storage, display, entertainment, communication, and IO devices. In a system employing Jini connection software, devices can be added or subtracted, and doing so may alter some of the capabilities of the system, but it will not alter its identity or basic usability. Jini technology requires a few things:

- an infrastructure which operates as a dynamically distributed system
- a common language and implementation that enables low-overhead communication between distributed objects
- a lookup service (which identifies objects that supply those services)
- an add-in protocol which is implemented on each device--we call this the discovery/join protocol
- a subtract-out mechanism--providing resilience when a device is unplugged--which is called leasing

4.2. Software Details

Jini technology ties together machines on which Java programming language objects are running, perhaps in different virtual machines. Jini technology enables such objects to work together as though they were on a single, very powerful computer: Such objects can be activated and tracked, can communicate with each other, and generally can be managed. A user can access all of the facilities on the collection of networked machines in a location-transparent fashion.

4.3. Infrastructure Functionality

Jini networking infrastructure provides resources for executing Java programming language objects, communication facilities between those objects, and the ability to find and exploit services on the network. By using Java Remote Method Invocation (RMI), Jini networking infrastructure provides communication between objects across device boundaries that enables those objects to work together. RMI enables activation of objects and the use of multicast to contact replicated objects, providing high availability and high reliance objects to be easily implemented in the Jini framework.

Jini technology provides a lookup service allowing services connected by the communication infrastructure to be found. Jini networking infrastructure provides a mechanism--called discovery/join--for Jini technology-enabled devices (such as disk drives, printers, and computers) to discover the appropriate lookup service and join into the overall system employing Jini technology. When a device joins a system employing Jini technology, its services are added to that lookup service. Symmetrically, when a Jini technology-enabled device leaves a system employing Jini technology (by being removed or by becoming unreliable), its services are deleted from the lookup service.

4.4. Technology Benefits

By providing a well-established distributed computing platform, which takes advantage of Java virtual machines running on a variety of platforms, users will see performance and reliability gains from applications designed to use Jini technology. Resources already provided in the Java programming language could see performance and reliability gains by being activated on unencumbered machines.

Jini technology provides the possibility to compose systems to meet specific requirements rather than relying on a general-purpose system. Particular services for a task or a group can be put together because Jini technology provides a low-impact way of customizing not only your software but also your hardware configuration. The services view of work enables devices and software to be managed uniformly.

Managing resources available on a system employing Jini technology is much simpler: Each Jini technology-enabled device has enough information stored on it to enable hot plugging. By simply plugging in a device, all of its Jini technology-related resources become available without intervention. Each device contains its own user interface for configuring and customizing the device.

The total cost of ownership of a computer system will decline, as fewer system administrators are needed. To be sure, in the short term a system administrator will still be required for even a medium-sized network, but the tasks he or she will be performing are those associated with the enterprise network, not routine resource maintenance. Further, systems for small business or departments and home use can be administered more reliably and at a much lower cost with Jini technology.

Jini technology begins to bring together the realms of computing and home networks including entertainment and personal/family management. Devices enabled by Jini technology can include computers, printers, scanners, and disks, but VCRs, DVD players, CD players, MIDI devices, cell phones, PDAs, the Web, and even broadcast receivers. As long as a device can attach to the network and can appear as a Java technology object, it can be a Jini technology-enabled device.

Finally, each new device and service that joins a community employing Jini technology increases the value of that community to the people using it, and each new device that becomes Jini technology-enabled increases the value of Jini technology in a spiral of increasing returns.

4.5. Components of a JINI System

In a running JINI system, there are three main players. There is a service, such as a printer, a toaster, a marriage agency, etc. There is a client, which would like to make use of this service. Thirdly, there is a lookup service (service locator), which acts as a broker/trader/locator between services and clients. There is an additional component,

and that is a network connecting all three of these, and this network will generally be running TCP/IP. (The JINI specification is fairly independent of network protocol, but the only current implementation is on TCP/IP.), as shown in figure:

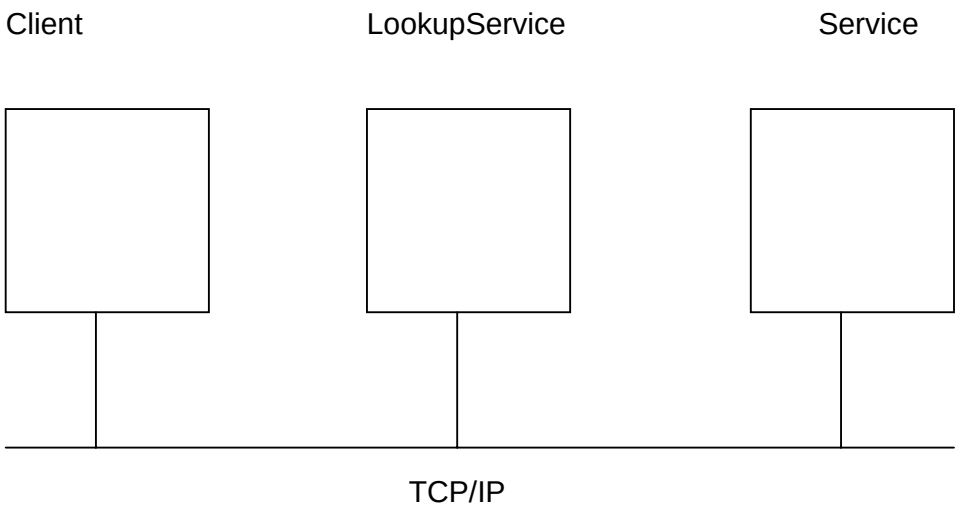


Figure 8 - Components of a JINI system

4.5.1. The Lookup service

A client locates a service by querying a lookup service (service locator). In order to do this, it must first locate such a service. On the other hand, a service must register itself with the lookup service, and in order to do so it must also first locate a service.
The initial phase of both a client and a service is thus discovering a lookup service. Such a service (or set of services) will usually have been started by some independent mechanism. The search for a lookup service can be done either by unicast or by multicast. In fact, the lookup service is just another JINI service, but it is one that is specialized to store services and pass them on to clients looking for them.

4.5.2. Reggie

Sun supplies a lookup service called Reggie as part of the standard JINI distribution. The specification of a lookup service is public, and in future we may expect to see other implementations of lookup services. There may be any number of these lookup services running in a network. A LAN may run many lookup services to provide

redundancy in case one of them crashes. Anybody can start a lookup service (depending on access permissions), but it is first of all not an easy job, and secondly it will usually be started by an administrator, or started at boot time.

Reggie requires support services to work: an HTTP server and an RMI daemon, rmid. If there is already an HTTP server running, this can be used, or a new one can be started. If you don't have access to an HTTP server (such as Apache), then there is a simple one supplied by JINI. This server is incomplete, and is only good for downloading Java class files - it cannot be used as a general-purpose Web server.

4.5.3. A JINI Service

The most important concept within the Jini architecture is that of a service. A service is an entity that can be used by a person, a program, or another service. A service may be a computation, storage, a communication channel to another user, a software filter, a hardware device, or another user. Two examples of services are printing a document and translating from one word-processor format to some other.

Members of a Jini system federate in order to share access to services. A Jini system should not be thought of as sets of clients and servers, or users and programs, or even programs and files. Instead, a Jini system consists of services that can be collected together for the performance of a particular task. Services may make use of other services, and a client of one service may itself be a service with clients of its own. The dynamic nature of a Jini system enables services to be added or withdrawn from a federation at any time according to demand, need, or the changing requirements of the workgroup using it.

Jini systems provide mechanisms for service construction, lookup, communication, and use in a distributed system. Examples of services include devices such as printers, displays, or disks; software such as applications or utilities; information such as databases and files; and users of the system.

Services in a Jini system communicate with each other by using a service protocol, which is a set of interfaces written in the Java programming language. The set of such protocols is open ended. The base Jini system defines a small number of such protocols, which define critical service interactions.

4.5.4. A JINI Client

A discovering entity that can retrieve a service (or remote reference to a service), registered with a discovered LUS and invoke the method of the service to meet the entity's requirements. A jini client can be a hardware device, a software or another jini service.

4.6. How JINI Works

4.6.1. Registering the Service with LUS

4.6.1.1. Discovering a LUS

A client locates a service by querying a lookup service (service locator). In order to do this, it must first locate such a service. On the other hand, a service must register itself with the lookup service, and in order to do so it must also first locate a service. The initial phase of both a client and a service is thus discovering a lookup service. The search for a lookup service can be done either by unicast or by multicast. In fact, the lookup service is just another Jini service, but it is one that is specialized to store services and pass them on to clients looking for them.

4.6.1.2. Discovery Protocols

The protocols used to locate a LUS are known as Discovery Protocols. Jini supports several useful protocols for different situations.

- **The Unicast Discovery Protocol** is used when an application or service already knows the particular lookup service it wishes to talk to. The Unicast Discovery Protocol is used to talk directly to a lookup service, which may not be a part of the local network, when the name of the lookup service is known
- **The Multicast Request Protocol** is used when an application or service first becomes active, and needs to find the "nearby" lookup services that may be active.

- **The Multicast Announcement Protocol** is used by lookup services to announce their presence. When a new lookup service that is part of an existing community starts up, any interested parties will be informed via the Multicast Announcement Protocol.

The end result of the discovery process is that proxies for the discovered lookup services are returned to the application doing the discovery.

4.6.1.3. LookupDiscoveryManager

An application (client or service) that wants to use a set of lookup services at fixed, known addresses and also to use whatever lookup services it can find by multicast can use the utility class LookupDiscoveryManager. Most of the methods of this class come from its interfaces:

```
package net.jini.discovery;
public class LookupDiscoveryManager implements DiscoveryManagement,
                                        DiscoveryGroupManagement,
                                        DiscoveryLocatorManagement {
 public LookupDiscoveryManager(String[] groups,
                    LookupLocator[] locators,
                    DiscoveryListener listener)
                       throws IOException;
}
```

This differs from LookupDiscovery and LookupLocatorDiscovery in that it insists on a DiscoveryListener in its constructor.

4.6.1.4. Registering Service with LUS(JoinManager)

A service needs to locate lookup services and register the service with them. Locating services can be done using the utility classes of ``Discovery Management''. As each lookup service is discovered, it then needs to be registered, and the lease maintained. The class JoinManager performs all of these tasks. There are two constructors

```
public class JoinManager {
    public JoinManager(Object obj,
                    Entry[] attrSets,
```

```
                ServiceIDListener callback,
                DiscoveryManagement discoverMgr,
                LeaseRenewalManager leaseMgr)
        throws IOException;
    public JoinManager(Object obj, Entry[] attrSets,  ServiceID serviceID,
                DiscoveryManagement discoverMgr,
                LeaseRenewalManager leaseMgr)  throws IOException;
}
```

The first of these is when the service is new and does not have a service id. A ServiceIDListener can be added which can note and save the id. The second form is used when the service already has an id. The other parameters are for the service and its entry attributes, a DiscoveryManagement object to set groups and unicast locators (typically this will be done using a LookupDiscoveryManager) and a LeaseRenewalManager.

4.6.2. Searching for and Using JINI Service

This section looks at how client search for and use the JINI service.

4.6.2.1. Client Lookup

The client tries to get a copy of the service into its own JVM. It goes through the same mechanism, as the service does, to get a registrar from the lookup service. It uses this to search for a service stored on that lookup service using the lookup() method:

```
        public Class ServiceRegistrar
        {
           public java.lang.Object lookup(ServiceTemplate tmpl)
           throws java.rmi.RemoteException;
public ServiceMatches lookup(ServiceTemplate tmpl,  int maxMatches)
                        throws java.rmi.RemoteException;
        }
```

The first of these methods just finds a service that matches the request. The second finds a set (upto the maxMatches) requested. If a client wishes to search for more than

one match to a service request from a particular lookup service, then it specifies the maximum number of matches it would like returned by the maxMatches parameter of the second for lookup() method.

So, we were talking about that the client goes through the same mechanism(i.e., trying to get a copy of the service into its own JVM) to get a registrar from the lookup service, but this time it does something different with this, which is to request the service object to be copied across to it as shown below:

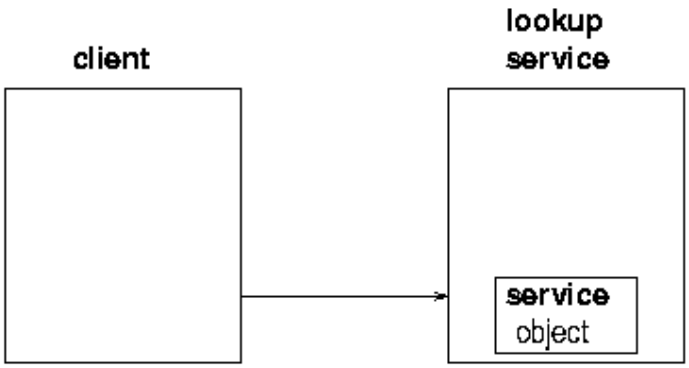


Figure 9 - Querying for a service locator

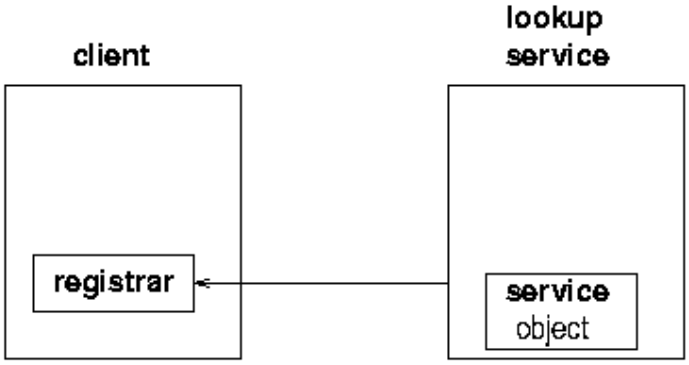


Figure 10 - Registrar returned

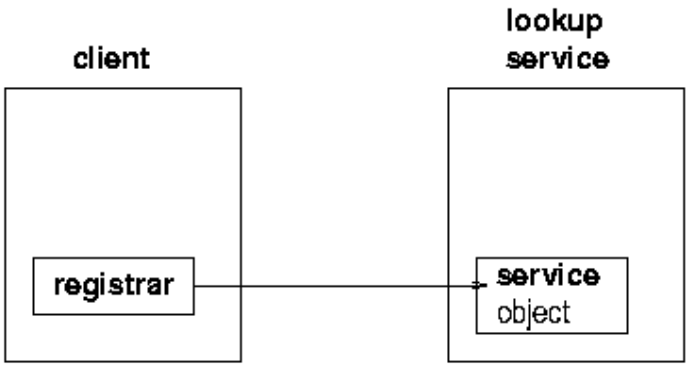


Figure 11 - Asking for a service

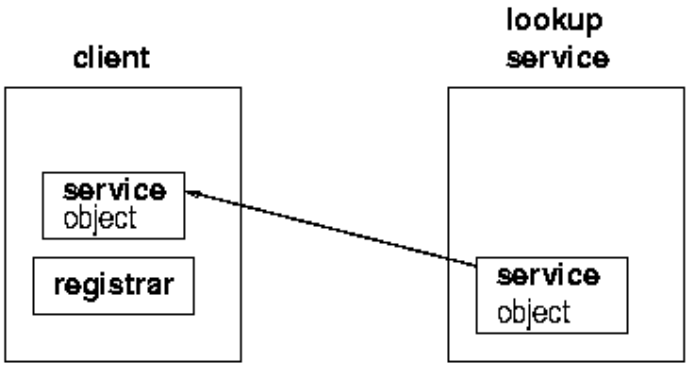


Figure 12 - Service returned

At this stage there is the original service object running back on its host. There is a copy of the service object stored in the lookup service, and there is a copy of the service object running in the client's JVM. The client can make requests of the service object running in its own JVM.

4.6.2.2. Proxies

Some services can be implemented by a single object, the service object. How does this work if the service is actually a toaster, a printer, or controlling some piece of hardware? By the time the service object runs in the client's JVM, it may be a long way away from its hardware. It cannot control this remote piece of hardware all by itself. In

this case, the implementation of the service must be made up of at least two objects, one running in the client and another distinct one running in the service provider.
The service object is really a proxy, which will communicate back to other objects in the service provider, probably using RMI. The proxy is the part of the service that is visible to clients, but its function will be to pass method calls back to the rest of the objects that form the total implementation of the service. There isn't a standard nomenclature for these server-side implementation objects. We shall refer to them in this document as the ``service backend'' objects.

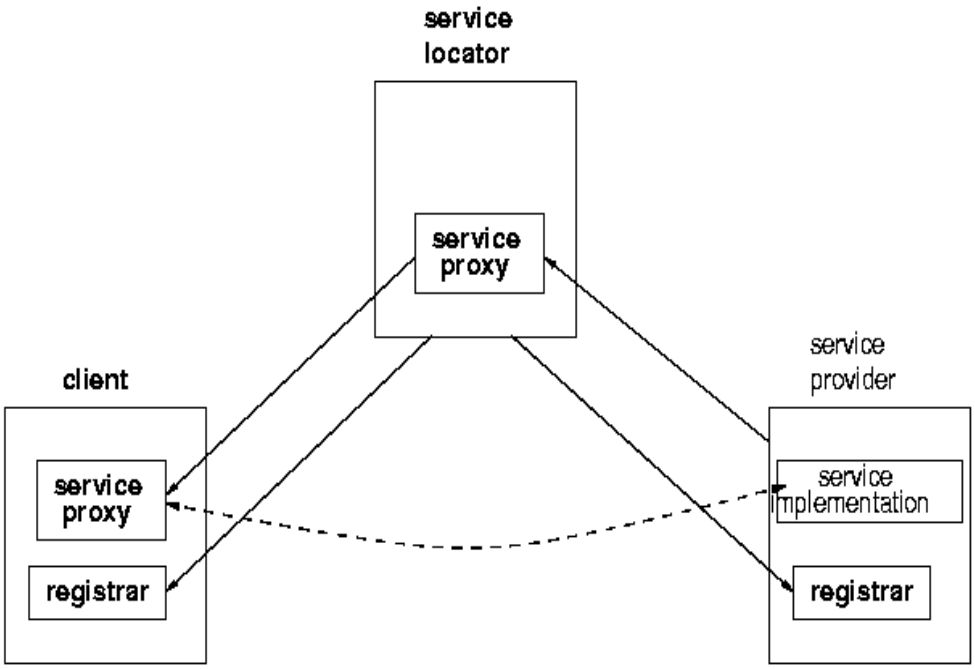


Figure 13 - A proxy service

4.6.2.3. Using the ServiceDiscoveryManager

The ServiceDiscoveryManager is a utility class that allows you to search for and discovery services of interest. The ServiceDiscoveryManager hides the entire notion of lookup services and ServiceRegistrar. The ServiceDiscovery-Manager takes care of finding lookup services, eliminating duplicate services, and the bookkeeping associated with discovery. As a developer, you need to deal only with services; most client applications use the ServiceDiscoveryManager class as their sole interface for service

location. This class offers broad support for clients who want to find services. It is a very flexible class that allows a number of different patterns of use to be supported.

4.6.3. Jini Leasing: Time-Based Resource Reservation

Jini provides communities that are stable, self-healing, and resilient in the face of (inevitable) network failures, machine crashes, and software errors. For this purpose Jini uses a technique called leasing. Leasing is based on the idea that, rather than granting access to a resource for an unlimited amount of time, the resource is “loaned” to some consumer for a fixed period of time. Jini leases require demonstrable proof-of-interest on the part of the resource consumer in order to continue to hold onto the resource.

Jini leases work much like leases in “real life.” The grantor of the lease may deny Jini Leases. The holder can renew them. Leases expire at a predetermined date unless they are renewed. They can be canceled early. Finally, leases can be negotiated, but, as in real life, the grantor has the final word on the terms of the lease that is offered.

Leases provide a consistent means to free unused or unneeded resources throughout Jini: If a service goes away, either intentionally or unintentionally, without cleaning up after itself, its leases eventually expire and the service will be forgotten. Leasing is used extensively by the lookup service and in other aspects of Jini, so it’s important to understand how leases work.

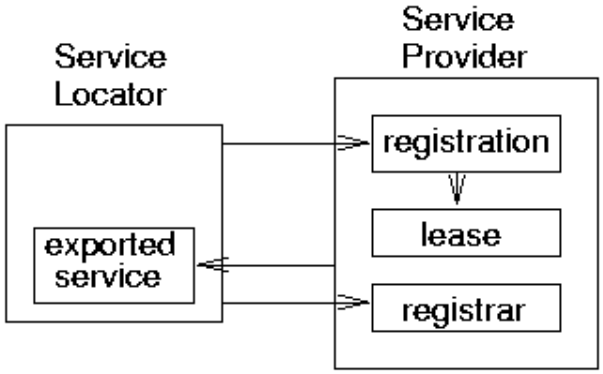


Figure 14 - Objects in a leased system

Lease Cancellation: A service can cancel its lease by using cancel(). The lease communicates back to the lease management system on the lookup service which cancels storage of the service.

Lease Expiration: When a lease expires, it does so silently. That is, the lease granter (the lookup service) will not inform the lease holder (the service) that it has expired. It is upto the service to call renew() before the lease expires if it wishes the lease to continue the service in milliseconds. Generally leases will be renewed and the manager will function quietly. However, the lookup service may decide not to renew a lease and will cause an exception to be thrown.

4.6.4. JINI Distributed Events

4.6.4.1. Event Models

Java has a number of event models, differing in various subtle ways. All of these involve an object generating an event in response to some change of state, either in the object itself (someone has changed a field, say), or in the external environment (a user has moved the mouse). At some earlier stage, a listener (or set of listeners) will have registered interest in this event and will have suitable methods called on them with the event as parameter.

4.6.4.2. Remote Events

Unlike the large number of event classes in the AWT and Swing (for example), Jini typically uses events of one type, the RemoteEvent or a small number of subclasses. The class has public methods

```
package net.jini.core.event;
public class RemoteEvent implements java.io.Serializable {
    public long getID();
    public long getSequenceNumber();
    public java.rmi.MarshalledObject getRegistrationObject();
}
```

Events in Beans and AWT convey complex object state information. Jini events avoid this, and convey just enough information to allow state information to be found if needed. A remote event is serializable and can be moved around the network to its listeners.

4.6.4.3. Event Registration

Jini does not say how to register listeners with objects that can generate events. This is unlike other event models in Java that specify methods such as

```
public void addActionListener(ActionListener listener);
```

for ActionEvent generators. What Jini does do is to specify a convenience class as a return value from this registration. This convenience class is

```
package net.jini.core.event;
import net.jini.core.lease.Lease;
public class EventRegistration implements java.io.Serializable {
   public EventRegistration(long eventID, Object source,
                            Lease lease, long seqNum);
   public long getID();
   public Object getSource();
   public Lease getLease();
   public long getSequenceNumber();
}
```

This return object contains information that may be of value to the object that registered a listener. Each registration will typically only be for a limited amount of time, and this information may be returned in the Lease object. If the event registration was for a particular type, this may be returned in the id field. A sequence number may also be given. The meaning of these may depend on the particular system - in other words, Jini gives you a class that is optional in use, and whose fields are not tightly specified. This gives you the freedom to choose your own meanings to some extent.

This means that as the programmer of a event producer, you have to define (and implement) methods such as

```
public EventRegistration addRemoteEventListener(RemoteEventListener listener);
```

There is no standard interface for this.

4.6.4.4. Monitoring Changes in Services

Services will start and stop. When they start they will inform the lookup services, and sometime after they stop they will be removed from the lookup services. But there are a lot of times when other services or clients will want to know when services start or are removed. For example: the editor that wants to know if a disk service has started so that it can save its file; the graphics display program that wants to know when printer services start up; the user interface for a camera that wants to track changes in disk and printer services so that it can update the ``Save'' and ``Print'' buttons.

A service registrar acts as a generator of events of type ServiceEvent which subclass from RemoteEvent. These events are generated in response to changes of state of services which match (or fail to match) a template pattern for services. This event type has three categories from the ServiceEvent.getTransition() method:

TRANSITION_NOMATCH_MATCH: a service has changed state so that whereas it previously did not match the template, now it does. In particular, if it didn't exist before now it does. This transition type can be used to spot new services starting. This transition can also be used to spot changes in the attributes of an existing registered service which are wanted: for example, an off-line printer can change attributes to being on-line, which now makes it a useful service

TRANSITION_MATCH_NOMATCH: a service has changed state so that whereas it previously did match the template, now it doesn't. This can be used to detect when services are removed from a lookup service. This transition can also be used to spot changes in the attributes of an existing registered service which are notwanted: for example, an on-line printer can change attributes to being off-line

TRANSITION_MATCH_MATCH: a service has changed state, but it matched both before and after. This typically happens when an Entry value changes, and is used to monitor changes of state such as a printer running out of paper, or a piece of hardware signalling that it is due for maintenance work

A client that wants to monitor changes of services on a lookup service must first create a template for the types of service it is interested in. A client that want to monitor all changes could prepare a template such as

ServiceTemplate templ = new ServiceTemplate(null, null, null); // or

ServiceTemplate templ = new ServiceTemplate(null, new Class[] {}, new Entry[] {}); // or

ServiceTemplate templ = new ServiceTemplate(null, new Class[] {Object.class}, null);

It then sets up a transition mask as a bit-wise OR of the three service transitions, and then calls notify() on the ServiceRegistrar object.

Chapter 5

5. LDAP

5.1. Introduction

The directory service in all its forms has quickly become a core component of eBusiness infrastructure development. And the surging number of disparate directories is likewise fueling the need for access and interoperability standards. Among these standards, Lightweight Directory Access Protocol (LDAP) enjoys broad industry acceptance as the protocol for deploying directory-based applications and solutions. Novell, the Sun-Netscape Alliance, Red Hat, Sun, Microsoft, Oracle, and IBM have all endorsed LDAP along with other leading ISV developers, consultants and system integrators. LDAP has evolved into the most popular open directory access mechanism in the recent times. Essentially this means that information can be stored in a hierarchical structure that is accessible from remote locations. Good examples of this are e-mail address lookup tables, white pages, and company structure information. LDAP is a client-server protocol for accessing a directory service. LDAP lets you "locate organizations, individuals, and other resources such as files and devices in a network, whether on the Internet or on a corporate intranet," and whether or not you know the domain name, IP address, or geographic whereabouts. An LDAP directory can be distributed among many servers on a network, then replicated and synchronized regularly. An LDAP server is also known as a Directory System Agent (DSA).

5.2. Overview of the Directories

Directories are special-purpose databases, usually containing categorized, more descriptive and attribute-based information to support frequent data retrieval and data

update. The Information in a directory is generally read much more often than it is written. Directories are tuned to give quick response to high-volume lookup or search operations. Directory updates are typically simple all-or-nothing changes. They may have the ability to replicate information widely in order to increase availability and reliability. The generic example of a directory would be a telephone directory or an address book. Different methods allow different kinds of information to be stored in the directory, place different requirement on how that information can be referenced, queried and updated, how it is protected from unauthorized access etc. The Directory service, a critical part of distributed computing, is the central point where network services, security services and applications form an integrated distributed computing environment.

5.3. LDAP Technology

LDAP is an emerging technology that can provide directory services to applications ranging from e-mail systems to distributed system management tools. LDAP is a simple directory data access protocol that supports a rich set of operations for a wide range of applications. LDAP is an open Internet standard, defined by the Internet Engineering Task Force (IETF). A number of implementations of LDAP are available, ranging from commercial to publicly available open-source products.

5.3.1. Terminologies

- **Entry.** In a generic sense, an entry is to a directory what a record or row is to a database.
- **Attributes.** An attribute is to a directory what a field is to a database.
- **Objects.** Objects in a directory are analogous to tables in a database. All entries of a particular object type will have the same kind of attributes.
- **Distinguished Name (DN).** The name used to uniquely (and globally) identify Bob is c=PAK, o=NUST, ou=Administration, cn=Babur Jamil. This is the distinguished name. Here an attribute, cn in this case, is chosen as the key that will represent the entry. The path leading to the entry with their values makes up the DN, thus the DN is the unique identifier of an entry.

- **Relative Distinguished Name (RDN).** Each level in the tree makes up a component of the DN to a particular node. Each of these components is called a RDN (Relative Distinguished Name).
- **Directory Information Tree (DIT).** The entire information tree of the directory itself is called the DIT (Directory Information Tree).
- **Schema.** The schema of an LDAP directory gives the layout of the information it contains and also how this information is grouped.

5.3.2. Components

- **LDAP server** is the server that LDAP clients interact with to obtain directory information.
- **LDAP data organization** is Back-end Database.
- **LDAP protocol** is the common language spoken by clients and servers when the clients access the directory.
- **LDAP clients** implemented using different vendor APIs and tools on different platforms are able to connect to the LDAP server.

5.3.3. Characteristics

- **Global Directory Service.** A well-designed LDAP directory allows users to access data that is uniquely identifiable on a global scale. To clarify this further, entities stored in an LDAP directory are unique in the sense that no two directory entities anywhere in the world will have the same identifier to access it.
- **Open Standard Interconnectivity**. The fact that LDAP can run on top of TCP/IP gives it the unique advantage of interconnectivity with machines similarly enabled.
- **Extremely fast Read Operations**
- **Relatively Static Data** .The data most commonly stored in the directory is not frequently subjected to change or modification.
- **Multi-Master replication**. Most leading directories offer multi-master replication, allowing writes and updates to occur on multiple servers. Therefore, even if

servers are unable to communicate for periods of time, operations can still occur locally and then be sent to other replicas once communication is restored.

- **Hierarchical**. The directory is capable of storing objects in a hierarchical fashion for organization and relationship.
- **Secure and Access Controlled Protocol**. LDAP is a secure protocol in that it makes use of authentication to ensure that transactions are secure. Authentication is used by the server to establish that the interacting client is who it claims to be. LDAP v3 uses the Simple Authentication and Security Layer (SASL).

5.3.4. Application

The key to deciding whether to choose an LDAP directory service is to understand what data can go into the LDAP directory. A few cases of what data can be represented using an LDAP directory and some examples of common directory applications are in order:

- The directory services can be white page services, yellow page services, or a query list of all the printers on the 6th floor. Queries with multiple constraints are also possible - list of all employees in the Engineering division working out of Pakistan with birthdays falling on April 24th.
- A very common LDAP application is seen in e-mail clients that auto-fill the e-mail address of the recipient when the name of the recipient is typed into the To: field.
- Several LDAP applications are actually gateways to other established services. For example, using a DNS to X.500 gateway, it is possible for LDAP-enabled clients to query a LDAP server for information residing on a legacy X.500 directory. Other examples can be e-mail to LDAP and finger to LDAP.
- Examples of data suited to reside in LDAP directories:
    - Employee phone book
    - Organizational charts
    - IT services information (for example, Domain names or IP addresses of servers)
    - E-mail addresses
    - URLs

- Binary data

5.3.5. Models

5.3.5.1. Informational Model

The informational model describes the informational units that go into the directory, that is, the entries and their types. The entries are the basic building blocks of the directory, which in turn are composed of attributes. The attributes of an entry are composed of an attribute type and one or more values. We design a schema for the directory that constrains the attributes and the type of attributes. Schemas are described in the LDIF (LDAP Data Interchange Format). LDIF is a standardized text-based format that describes directory entries. Using the LDIF format, data from one directory can be exported to another regardless of the actual format the two directories use for their data stores. Let's take example of a typical address book that we would maintain online in our LDAP directory. The corresponding LDIF representation for this directory is as under.

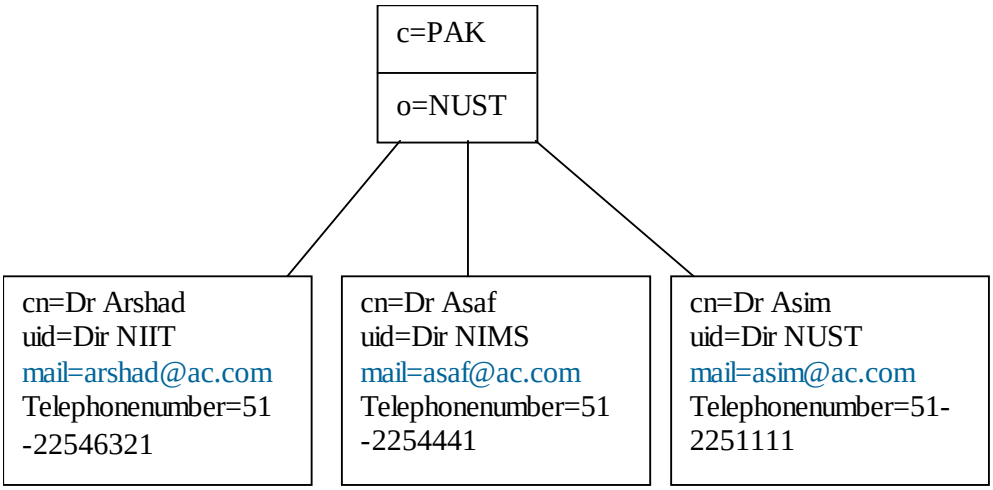


Figure 15 – Directory Information Tree

The top level entry which represents the node with o=NUST is:

dn: o= NUST, c= PAK

objectclass=top

objectclass=organization

o=NUST

Taking example of one of the nodes:

dn: mail=arshad@ac.com, o= NUST, c= PAK

cn: Dr Arshad

objectclass: top

objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson

uid: Dir NIIT

mail: arshad@ac.com

Telephonenumber: 51-22546321

The distinguished name (dn) label within this entry is an attribute. So are the objectclass and organization (o) labels. DN attribute is the unique identifier for an entry and it traces the path of the entry from the top-level root. The DN consists of comma-separated RDNs and usually the left-most RDN is an attribute of the entry itself.

5.3.5.2. Naming Model

The LDAP naming model specifies how the directory data is organized. In the LDAP data organization there is a root entry that has other entries below it that in turn might contain sub-entries. Let's look at the entry for Dr Arshad once again. The DN for this entry would be mail=arshad@ac.com, o=NUST, c=PAK. “mail=arshad@ac.com” is an RDN, this means that there cannot be another entry under o=NUST,c=PAK with the value arshad@ac.com for the mail attribute, if not the uniqueness is lost. Note that by default, while the names of the attributes are case- sensitive, the values are not; in this case cn is case-sensitive while Dr Arshad is not.

5.3.5.3. Functional Model

This model defines the operations that can be performed on the data stored in the directory. It also dictates which clients the users can access and which parts of the directory they can change. There are some basic operations that can be divided into three of the following categories:

5.3.5.3.1. Query Operation

These are essentially search and compare operations, which allow us to search the directory and compare two or more entries and their attributes. We can use search filters to perform the search and compare operations. The search-filter is a regular expression with the names of attributes and operators that might match the entries with the appropriate attributes.

For example, in the some organizational chart, a search filter of the form:

- (cn=*a*) (ou=SRoom) will match all entries of employees in the Server Room with the letter a in their names.
- (cn=*) will match all entries with a cn attribute.

5.3.5.3.2. Update Operations

Add. Entries can be added to the pre-existing set of entries in the directory, as long as they conform to the schema of the directory. The objects have some mandatory fields that are to be filled and the rest are optional. When an entry is added to the directory, its DN must be specified so that the LDAP server will know where to graft the entry into the tree. The client must have sufficient privileges to perform an add operation.

Delete. Deleting an entry from the directory.

Modify. Supplying the LDAP server with the DN of the entry and the set of attributes that need to be modified can modify entries.

5.3.5.3.3. Authentication Operation

Bind. This operation has a DN and a set of authentication credentials, which it supplies to the server.

Unbind. The unbind operator has no arguments to it. It discards authentication credentials and terminates the underlying network connection.

5.3.5.4. Security Model

An LDAP service provides a generic directory service. All LDAP servers have some system in place for controlling who can read and update the information in the directory.

To access the LDAP service, the LDAP client first must authenticate itself to the service. That is, it must tell the LDAP server who is going to be accessing the data so that the server can decide what the client is allowed to see and do. If the client authenticates successfully to the LDAP server, then when the server subsequently receives a request from the client, it will check whether the client is allowed to perform the request. This process is called access control.

Another security aspect of the LDAP service is the way in which requests and responses are communicated between the client and the server. Many LDAP servers support the use of secure channels to communicate with clients, for example to send and receive attributes that contain secrets, such as passwords and keys. LDAP servers use SSL for this purpose. The security model specifies how the contents of the directory can be protected from unauthorized access. It also specifies the scope of access for the clients, that is, it specifies which clients can access which parts of the directory tree and whether they can perform update or interrogation operations or both on that part of the tree.

Different versions of the LDAP support different types of authentication. The LDAP v2 defines three types of authentication: anonymous, simple (clear-text password), and Kerberos v4. The LDAP v3 supports anonymous, simple, and SASL authentication. SASL is the Simple Authentication and Security Layer. Since LDAPv3, SASL handles authentication and security it is merely a standard way for plugging in different authentication protocols that do the actual work of authentication and enforcement of security.

5.3.5.4.1. Simple Authentication

Simple authentication consists of sending the LDAP server the fully qualified DN of the client (user) and the client's clear-text password. This mechanism has security problems because the password can be read from the network.

If you supply an empty string, an empty byte/char array, or null, then the authentication mechanism will be "none". This is because the LDAP requires the password to be nonempty for simple authentication.

5.3.5.4.2. SASL

The LDAP v3 protocol uses the SASL to support pluggable authentication. This means that the LDAP client and server can be configured to negotiate and use possibly nonstandard and/or customized mechanisms for authentication, depending on the level of protection desired by the client and the server.

5.3.6. Advance Features of LDAP

Let's discuss some features which LDAP supports but are seldom used except by administrators or advanced users.

5.3.6.1. Asynchronous Operations

LDAP supports asynchronous operations on the directory. Asynchronous operations are operations that do not block.

5.3.6.2. Replication

In some large installations there would be producer and consumer LDAP servers. The updates are always done to the producer servers and they are periodically replicated with the consumer servers. The clients always access the consumer servers. The advantage of this is that the client operations are fast since they talk to servers that are not bogged down by the performance overhead associated with updates.

5.3.6.3. Referrals

The referral service allows LDAP servers to distribute, de- centralize, and load-balance their processing. In the simple case of a referral, the LDAP server may choose to redirect the client to another LDAP server for a piece of information that the client requested. This allows for de-centralization because individual organizations within a company need to maintain only data specific to them and other servers can redirect queries to them that are specific to each of these organizational servers.

5.3.6.4. Security

LDAP directories may store sensitive information such as Social Security Numbers, passwords, private keys, and other sensitive information. The protocol provides for safe transaction of such sensitive data by providing SASL that is flexible enough to accommodate various underlying encryption or certification schemes. Further, LDAP enforces access control for the operations that various users can perform on the directory.

5.4. Directory Server

Because LDAP defines a client/server data management system (DMS), a working LDAP system must have an LDAP server and client. These days, vendors largely distinguish themselves by the robustness, performance, and manageability of their server. In this project iPlanet Directory Server v5.1, the latest available version, has been used, a Sun-Netscape Alliance product. iPlanet is the brand for Alliance products. iPlanet Directory Server 5.1 is a powerful and scalable distributed directory server based on the industry-standard Lightweight Directory Access Protocol (LDAP).

5.4.1. Directory Deployment

A directory service consists of at least one instance of Directory Server and one or more directory client programs. Client programs can access names, phone numbers, addresses, and other data stored in the directory. When you install Directory Server, the following components are installed on your machine:

- A server front-end responsible for network communications which communications with directory client programs. Directory Server functions as a service on Windows NT and Windows 2000. Multiple client programs can speak to the server in LDAP. They can communicate using LDAP over TCP/IP.
- Plug-ins for server functions, a plug-in is a way to add functionality to the core server. For example, a database is a plug-in.
- The directory tree, also known as a directory information tree or DIT, mirrors the tree model used by most file systems, with the tree's root, or first entry,

appearing at the top of the hierarchy. At installation, Directory Server creates a default directory tree.

5.4.1.1. Data Planning

Data is the key to your directory architecture. The type of data in your directory determines how you structure the directory, to whom you allow access to the data, and how this access is requested and granted. This is done through site survey. A site survey is a formal method for discovering and characterizing the contents of your directory. The site survey consists of Identifying the applications that use your directory, identifying data sources, characterize the data your directory needs to contain, determine what objects should be present in your directory etc. Some types of data are better suited to your directory than others. Ideal data for a directory has some of the following characteristics:

- It is expressible in attribute-data format (for example, surname=nemati).
- It will be accessed from more than one physical location
- Directory Server is excellent for managing large quantities of data that client applications read and occasionally write, but it is not designed to handle large, unstructured objects, such as images or other media. These objects should be maintained in a file system. However, your directory can store pointers to these kinds of applications through the use of FTP, HTTP, or other types of URL.

5.4.1.2. Designing Schema

Your directory schema describes the types of data you can store in your directory. During schema design, you map each data element to an LDAP attribute, and gather related elements into LDAP object classes. Well-designed schema helps maintain the integrity of the data you store in your directory. During schema design, you select and define the object classes and attributes used to represent the entries stored by Directory Server. Schema design involves the following steps:

- Choosing predefined schema elements to meet as many of your needs as possible.

- Extending the standard Directory Server schema to define new elements to meet your remaining needs.
- Planning for schema maintenance.

It is best to use existing schema elements defined in the standard schema provided with Directory Server. Choosing standard schema elements helps ensure compatibility with directory-enabled applications. Attributes hold specific data elements such as a name or a fax number. Directory Server represents data as attribute-data pairs, a descriptive attribute associated with a specific piece of information. So, an entry for a person named Babur Jamil has the attribute-data pair like: cn: Babur Jamil

Each directory entry belongs to one or more object classes. Once you place an object class identified in your schema on an entry, you are telling the directory server that the entry can have a certain set of attribute values and must have another, usually smaller, set of attribute values. While this schema meets most directory needs, you may need to extend it with new object classes and attributes to accommodate the unique needs of your directory.

5.4.1.3. Designing Directory Tree

Your directory tree provides a way to refer to the data stored by your directory. The types of information you store in your directory, the physical nature of your enterprise, the applications you use with your directory, and the types of replication you use shape the design of your directory tree. The structure of your directory tree follows the hierarchical LDAP model. Your directory tree provides a way to organize your data, for example, by group, by people, or by place. It also determines how you partition data across multiple servers. The directory tree design process involves the following steps:

- **Choosing a suffix to contain your data**. The suffix is the name of the entry at the root of your tree, below which you store your directory data. Your directory can contain more than one suffix. You may choose to use multiple suffixes if you have two or more directory trees of information that do not have a natural common root. By default, the standard iPlanet Directory Server deployment contains multiple suffixes, one for storing data and the others for data needed by internal directory operations. All entries in your directory should be located below a common base entry, the root suffix.

- Determining the hierarchical relationship among data entries.
- Naming the entries in your directory tree hierarchy.

5.4.2. Directory Administration

The Directory Server runs as the ns-slapd process or service on your machine. The server manages the directory databases and responds to client requests. iPlanet Console is the graphical interface to the Administration Server. You can perform most Directory Server administrative tasks from the Directory Server Console. You can also perform administrative tasks manually by editing the configuration files. This includes:-

- Creating Entries
    - Creating root entries
    - Creating Directory entries
- Configuring Directory Databases

5.5. JNDI

A directory service provides access to diverse kinds of information about users and resources in a network environment. It uses a naming system for purpose of identifying and organizing directory objects to represent this information. JNDI is used to communicate with LDAP server(s). Developers only worry about only one particular protocol (LDAP) and API (JNDI) and nothing more. A directory object can have associated with it attributes. An attribute has an identifier and a set of values. A directory object provides an association between attributes and values. Thus, a directory service enables information to be organized in a hierarchical manner to provide a mapping between human understandable names and directory objects. In the following diagram, client is not worried about the environment being used but he only interacts with JNDI, which interacts in turn with LDAP interfaces.

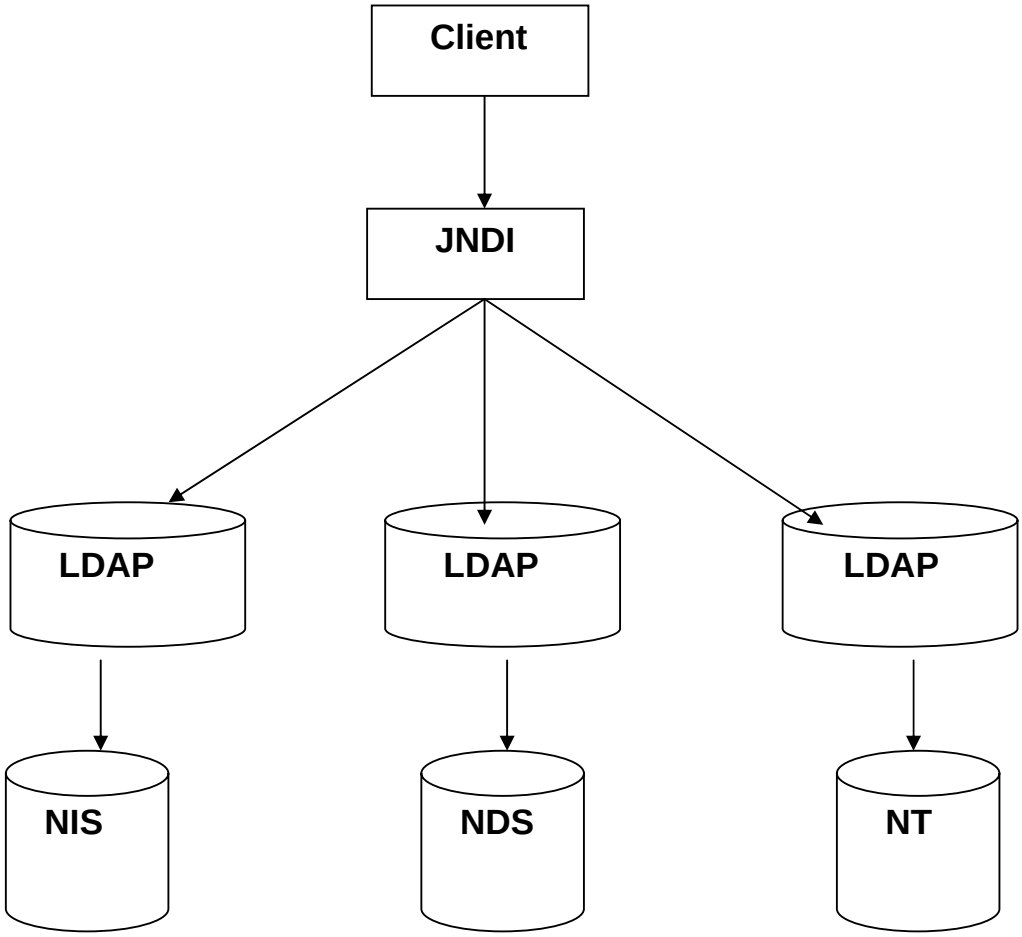


Figure 16 – JNDI Deployment

5.5.1. Naming

A fundamental facility in any computing system is the naming service – the means by which names are associated with objects, and by which objects are found given their names. The computing environment of an enterprise typically consists of several naming services. There are naming services that provide contexts for naming common entities in an enterprise such as organizations, physical sites, human users and computers. Naming services are also incorporated in applications offering services such as file service, mail service, printer service, and so on. The primary function of a naming system is to map names to objects. The objects can be of any type. A naming system is a connected set of contexts of the same type (having the same naming

convention) and providing the same set of operations with identical semantics. A **namespace** is the set of all names in a naming system. **Naming convention**. Every name is generated by a set of syntactic rules called a naming convention. A **context** is an object whose state is a set of bindings with distinct atomic names. Every context has an associated naming convention. A context provides a lookup (resolution) operation that returns the object.

5.5.2. Architecture

The JNDI architecture consists of the JNDI API and the JNDI SPI. The JNDI API allows Java applications to access a variety of naming and directory services. The JNDI API is contained in two packages: javax.naming for the naming operations, and javax.naming.directory for directory operations. The JNDI service provider interface is contained in the package javax.naming.spi. Context defines basic operations such as adding a name-to-object binding, looking up the object bound to a specified name etc. There are two main settings in which JNDI is used: in Java applications and applets.

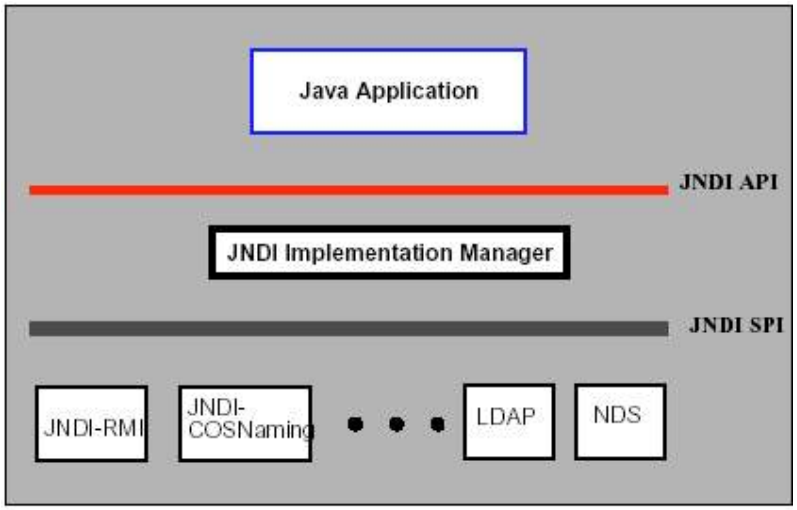


Figure 17 – JNDI Architecture

Because schemas can be expressed as an information tree, it is natural to use for this purpose the naming and directory interfaces already defined in JNDI. This is powerful because the schema part of a namespace is accessible to applications in a uniform

way. A browser, for example, can access information in the schema tree just as though it were accessing any other directory objects. JNDI does not define a security model or a common security interface for accessing naming and directory servers.

Chapter 6

6. Object Oriented Design

6.1. Use Case Modeling

6.1.1. List of Actors

- Client/User
- Administrator

6.1.2. Use Case List

- Define User Requirements
- Specify Resource Attributes

6.1.3. Use Case Specifications

6.1.3.1. Define User Requirements

This use case starts when user selects the option of "Define User Requirement" on GUI available for this purpose. On clicking this, another GUI appears containing various Text Fields. User enters the values in these Text Fields specifying his/her requirements. This use case ends when user clicks "Ok" button on GUI.

6.1.3.2. Specify Resource Attributes

This use case starts when user selects the option of "Specify Resource Attributes" on GUI available for this purpose. On clicking this, another GUI appears containing various Text Fields. User enters the values in these Text Fields specifying the attributes of the generated resource. Specifying these attributes and then clicking "Ok" button on GUI will result in creation of a new Template.

6.1.4. Use Case Diagrams

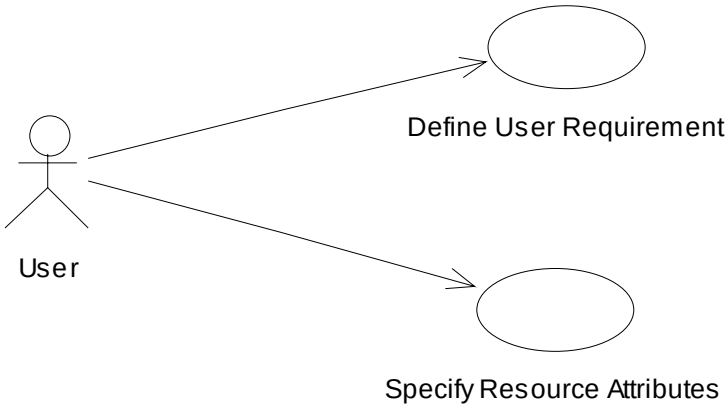


Figure 18 – Use Case Diagram

6.2. Class Modeling

6.2.1. Problem Domain Object List (PDOL)

- Tangibles
  - Resource Generator
  - Resource Consumer

- Events
    - User specifies hid demand.
    - User generates a resource.
    - Resource is parsed.
    - Resource is passed to the Notification Service.
    - Notification service send template along with notification to remote Notification service.
    - Remote notification passes the template PA.
    - PA stores the template in LDAP.
    - PA passes the template to KA.
    - KA compares the incoming template with its knowledge base.
    - KA passes the Tid to PA.
    - PA generates the FA.
    - FA goes to the remote PA.
    - FA passes the Tid to TA.
    - LDAP returns MappingInfo to TA.
    - TA passes the MappingInfo and DMI to TS.
    - TS starts transfer of resource.
    - TS sends the completion to TA.
    - FA returns.
- Identification Of Classes
    - Resource Agent
    - Home Agent
    - Template
    - Notification Agent
    - Event Generator
    - Event Consumer
    - Event Notifier
    - Event Thread
    - Print Service ID
    - Template Interface Remote
    - Display Message
    - Platform Address

- Mapping Info
- Transaction Agent
- Transfer Initiator
- File Transfer
- File Downloader
- File Download
- Transfer Interface
- Platform Agent
- Knowledge Agent
- DF (Directory Facilitator)
- Agent Management Service (AMS)
- Agents Progress Display
- LDAP Agent
- Jadd
- Jsearch
- Document1
- Jdel
- Lserver
- Fetch Agent
- Jini Agent
- Monitor Agent

6.2.2. Class Relationship Diagram (CRD)

The CRDs of the entire application are shown on subsequent pages. Due to space constraints the first diagram just shows the class names and their interactions. The details of these classes are covered in the subsequent diagrams. The class relations and collaborations (CRC cards) are presented in the next section.

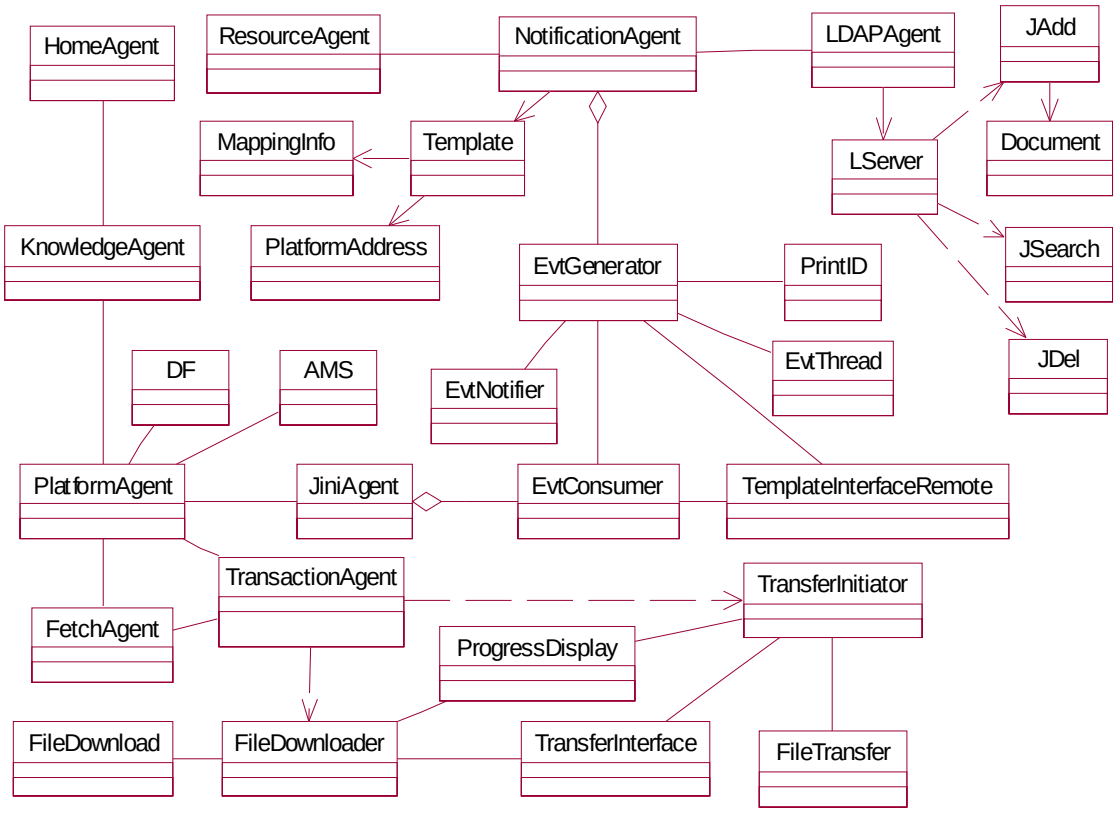
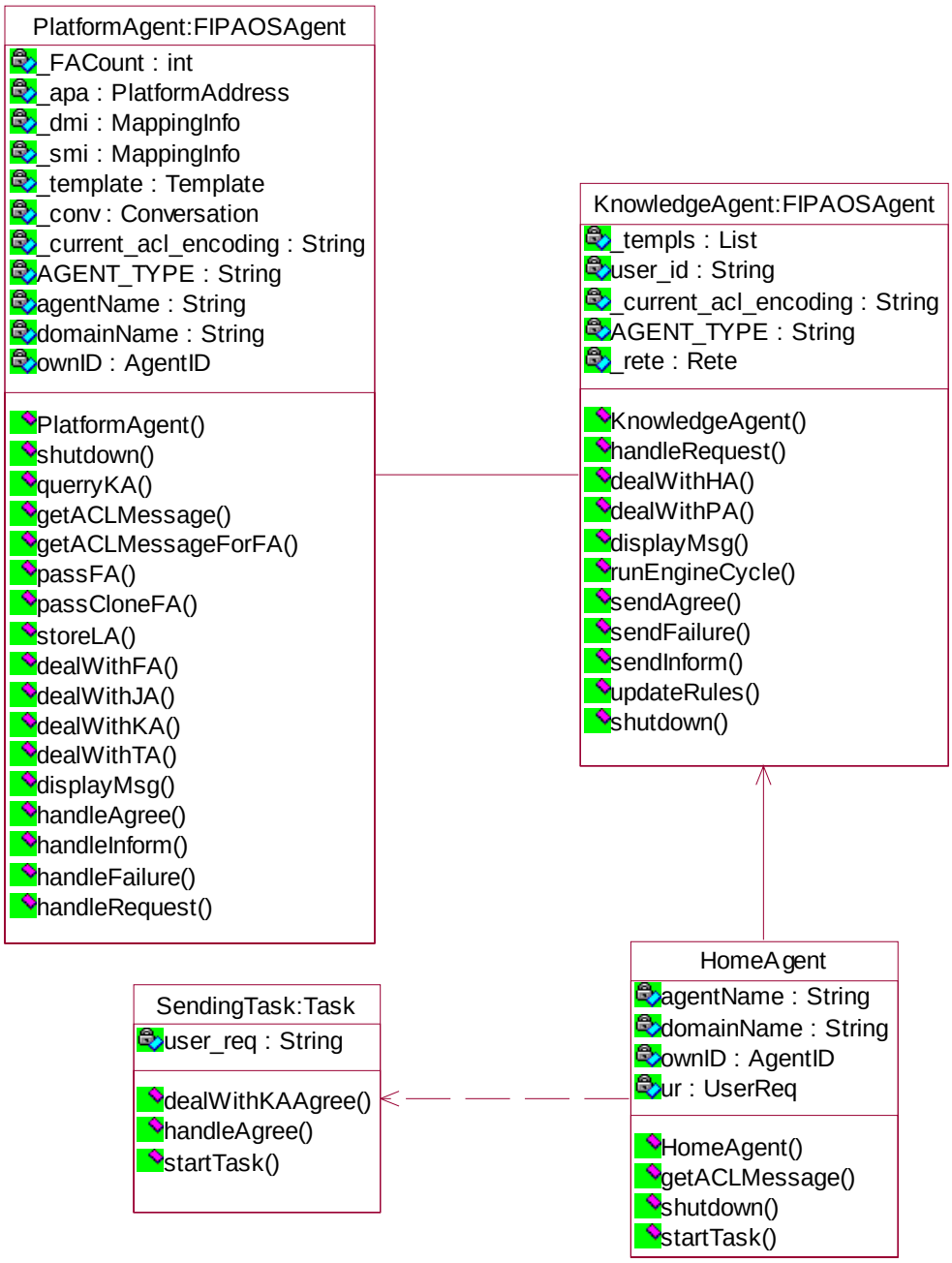


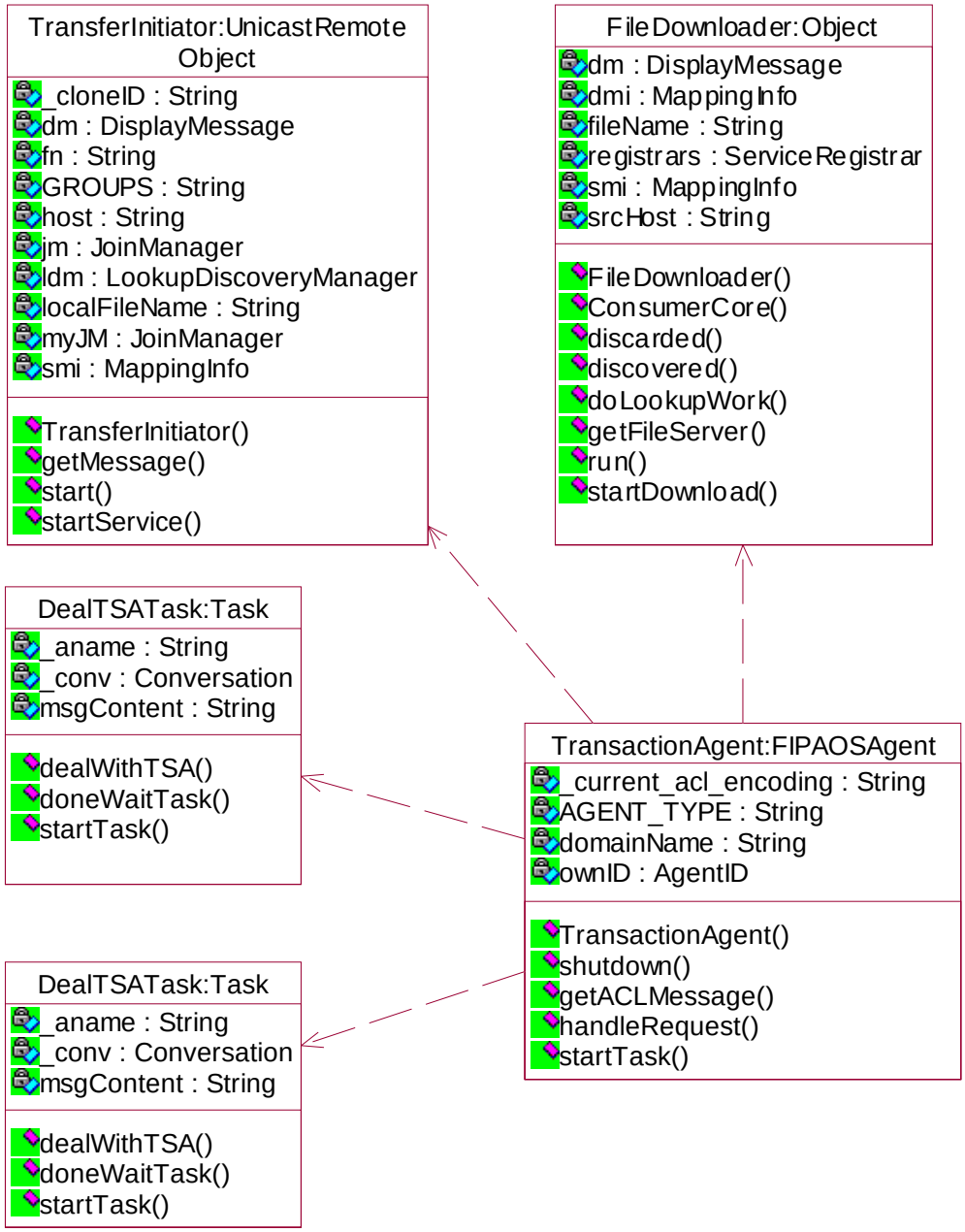
Figure 19 – CRD for KMS


PlatformAgent:FIPAOSAgent
_FACount : int
_apa : PlatformAddress
_dmi : MappingInfo
_smi : MappingInfo
_template : Template
_conv : Conversation
_current_acl_encoding : String
AGENT_TYPE : String
agentName : String
domainName : String
ownID : AgentID
PlatformAgent()
shutdown()
querryKA()
getACLMessage()
getACLMessageForFA()
passFA()
passCloneFA()
storeLA()
dealWithFA()
dealWithJA()
dealWithKA()
dealWithTA()
displayMsg()
handleAgree()
handleInform()
handleFailure()
handleRequest()
KnowledgeAgent:FIPAOSAgent
_templs : List
user_id : String
_current_acl_encoding : String
AGENT_TYPE : String
_rete : Rete
KnowledgeAgent()
handleRequest()
dealWithHA()
dealWithPA()
displayMsg()
runEngineCycle()
sendAgree()
sendFailure()
sendInform()
updateRules()
shutdown()
HomeAgent
agentName : String
domainName : String
ownID : AgentID
ur : UserReq
HomeAgent()
getACLMessage()
shutdown()
startTask()
SendingTask:Task
user_req : String
dealWithKAAgree()
handleAgree()
startTask()


ResourceAgent:FIPAOSAgent
_current_acl_encoding : String
_serializedTemplate : Object
AGENT_TYPE : String
agentName : String
domainName : String
ownID : AgentID
strTempl : String
_template : Template
ur : UserReq
ResourceAgent()
shutdown()
getACLMessage()
startTask()
NotificationAgent:FIPAOSAgent
_current_acl_encoding : String
_serializedTemplate : Object
_template : Template
AGENT_TYPE : String
domainName : String
myEvent : EvtGenerator
ownID : AgentID
NotificationAgent()
shutdown()
createTemplate()
displayTemplate()
getACLMessage()
handleRequest()
startTask()
storeLA()
LDAPAgent:FIPAOSAgent
_current_acl_encoding : String
AGENT_TYPE : String
ownID : AgentID
LDAPAgent()
retSmi()
shutdown()
dealWithNA()
dealWithPA()
dealWithTA()
handleRequest()
startTask()
JiniAgent
_template : Template
_serializedTemplate : Object
_current_acl_encoding : String
AGENT_TYPE : String
JiniAgent()
shutdown()
getACLMessage()
serialize()
startTask()

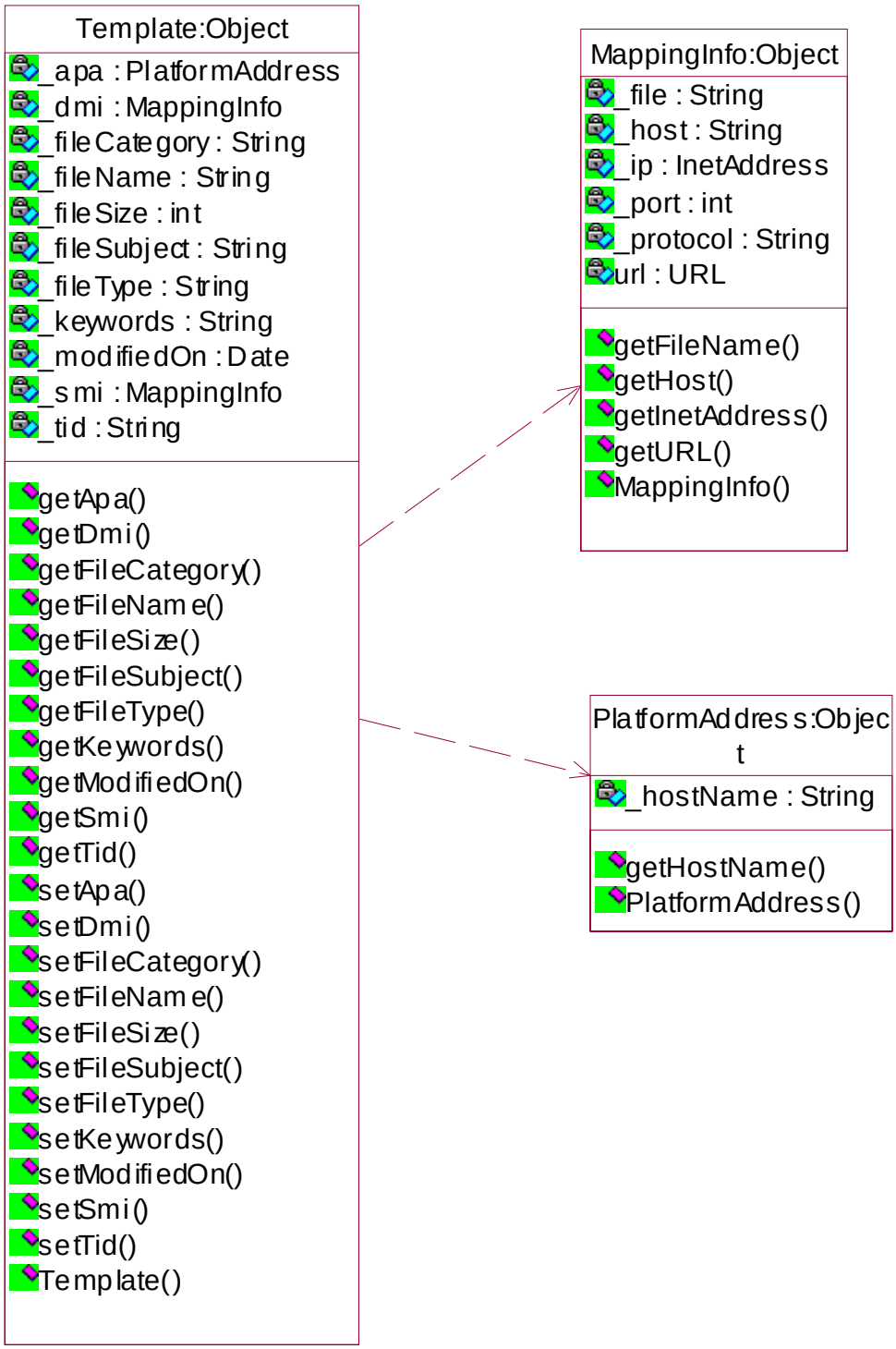

NotificationAgent:FIPAOSAgent
_current_acl_encoding : String
_serializedTemplate : Object
_template : Template
AGENT_TYPE : String
domainName : String
myEvent : EvtGenerator
ownID : AgentID
NotificationAgent()
shutdown()
createTemplate()
displayTemplate()
getACLMessage()
handleRequest()
startTask()
storeLA()
JiniAgent
_template : Template
_serializedTemplate : Object
_current_acl_encoding : String
AGENT_TYPE : String
JiniAgent()
shutdown()
getACLMessage()
serialize()
startTask()
EvtGenerator:UnicastRemotObject
dm : DisplayMessage
evtSeqNum : long
export : RemoteObject
GROUPS : String
index : int
jm : JointManager
ldm : LookupDiscoveryManager
listener : RemoteEventListener
myJM : JoinManager
myList : EventListenerList
templ : Template
EvtGenerator()
addRemoteListener()
fireRemoteEvent()
getIndex()
getTemplate()
startService()
showMessage()
EvtConsumer:UnicastRemote Object
dm : DisplayMessage
evtId : long
matchevtId : long
matchTid : String
registrars : ServiceRegistrar
tid : String
EvtConsumer()
ConsumerCore()
discarded()
discovered()
doEventReg()
doLookupWork()
notify()

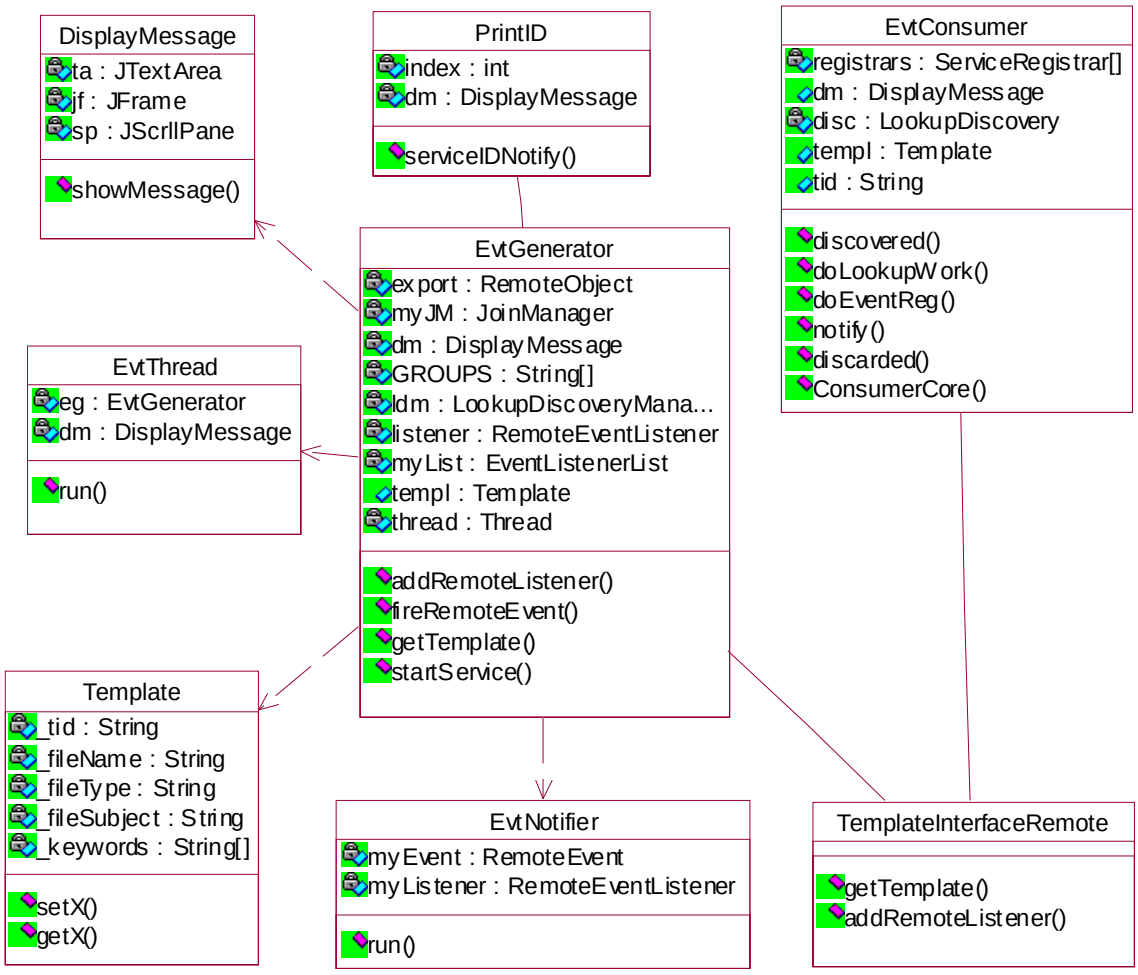

PlatformAgent:FIPAOSAgent
_FACount : int
_apa : PlatformAddress
_dmi : MappingInfo
_smi : MappingInfo
_template : Template
_conv : Conversation
_current_acl_encoding : String
AGENT_TYPE : String
agentName : String
domainName : String
ownID : AgentID
PlatformAgent()
shutdown()
querryKA()
getACLMessage()
getACLMessageForFA()
passFA()
passCloneFA()
storeLA()
dealWithFA()
dealWithJA()
dealWithKA()
dealWithTA()
displayMsg()
handleAgree()
handleInform()
handleFailure()
handleRequest()
FetchAgent:FIPAOSAgent
_current_acl_encoding : String
msgContent : String
domainName : String
homeTS : String
orgName : String
ownID : AgentID
AGENT_TYPE : String
FetchAgent()
dealWithPA()
dealWithFA()
dealWithFAInform()
dealWithFAState()
dealWithPAAddress()
dealWithTSA()
doneWaitTask()
startTask()
displayMsg()
getACLMessage()
handleRequest()
handleInform()
handleOther()
shutdown()
JiniAgent
_template : Template
_serializedTemplate : Object
_current_acl_encoding : String
AGENT_TYPE : String
JiniAgent()
shutdown()
getACLMessage()
serialize()
startTask()
TransactionAgent:FIPAOSAgent
_current_acl_encoding : String
AGENT_TYPE : String
domainName : String
ownID : AgentID
TransactionAgent()
shutdown()
getACLMessage()
handleRequest()
startTask()

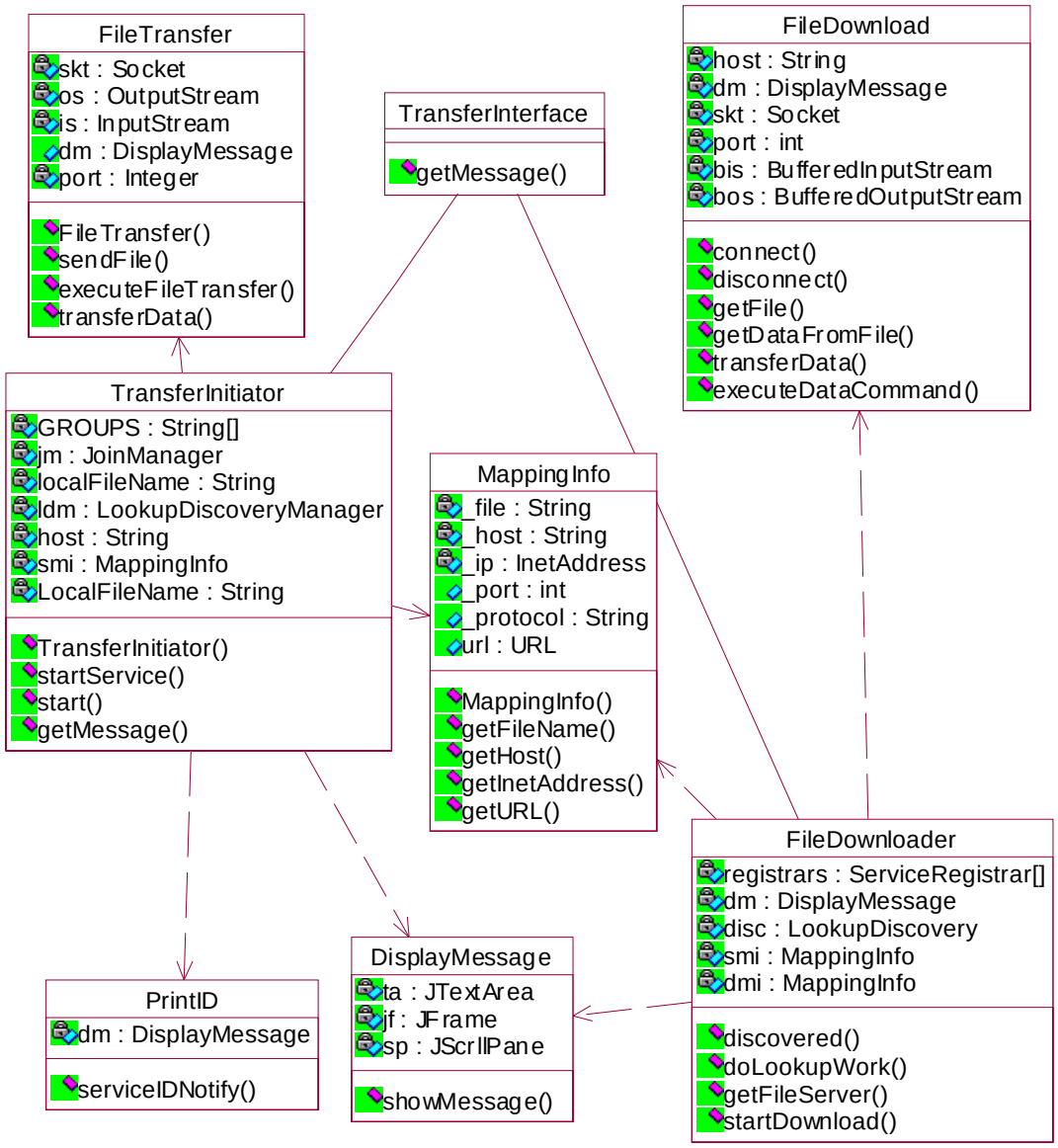

TransferInitiator:UnicastRemote Object
_cloneID : String
dm : DisplayMessage
fn : String
GROUPS : String
host : String
jm : JoinManager
ldm : LookupDiscoveryManager
localFileName : String
myJM : JoinManager
smi : MappingInfo
TransferInitiator()
getMessage()
start()
startService()
FileDownloader:Object
dm : DisplayMessage
dmi : MappingInfo
fileName : String
registrars : ServiceRegistrar
smi : MappingInfo
srcHost : String
FileDownloader()
ConsumerCore()
discarded()
discovered()
doLookupWork()
getFileServer()
run()
startDownload()
DealTSATask:Task
_aname : String
_conv : Conversation
msgContent : String
dealWithTSA()
doneWaitTask()
startTask()
TransactionAgent:FIPAOSAgent
_current_acl_encoding : String
AGENT_TYPE : String
domainName : String
ownID : AgentID
TransactionAgent()
shutdown()
getACLMessage()
handleRequest()
startTask()
DealTSATask:Task
_aname : String
_conv : Conversation
msgContent : String
dealWithTSA()
doneWaitTask()
startTask()

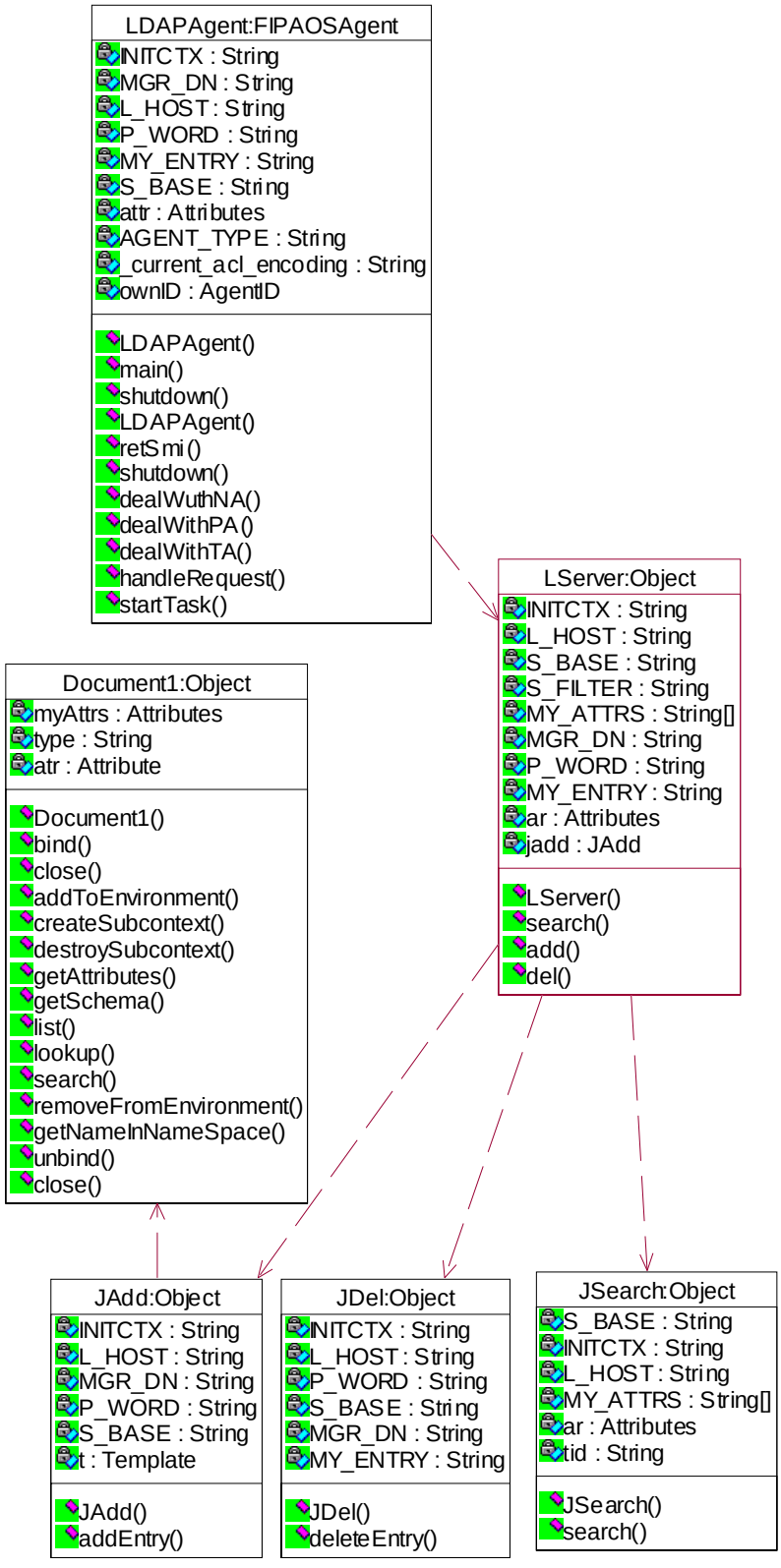

PlatformAgent
_FACount : int
_apa : PlatformAddress
_dmi : MappingInfo
_smi : MappingInfo
_template : Template
_conv : Conversation
_current_acl_encoding : String
AGENT_TYPE : String
agentName : String
domainName : String
ownID : AgentID
PlatformAgent()
shutdown()
querryKA()
getACLMessage()
getACLMessageForFA()
passFA()
passCloneFA()
storeLA()
dealWithFA()
dealWithJA()
dealWithKA()
dealWithTA()
displayMsg()
handleAgree()
handleInform()
handleFailure()
handleRequest()
LDAPAgent:FIPAOSAgent
_current_acl_encoding : String
AGENT_TYPE : String
ownID : AgentID
LDAPAgent()
retSmi()
shutdown()
dealWithNA()
dealWithPA()
dealWithTA()
handleRequest()
startTask()
NotificationAgent:FIPAOSAgent
_current_acl_encoding : String
_serializedTemplate : Object
_template : Template
AGENT_TYPE : String
domainName : String
myEvent : EvtGenerator
ownID : AgentID
NotificationAgent()
shutdown()
createTemplate()
displayTemplate()
getACLMessage()
handleRequest()
startTask()
storeLA()
TransactionAgent:FIPAOSAgent
_current_acl_encoding : String
AGENT_TYPE : String
domainName : String
ownID : AgentID
TransactionAgent()
shutdown()
getACLMessage()
handleRequest()
startTask()


MonitorAgentGUI:JFrame
_active : List
_all : List
_ma : MonitorAgent
_owner_name : String
_table_model : PAGTableModel
MonitorAgentGUI()
update()
exitForm()
initComponents()
MonitorAgent:FIPAOSAgent
_agents : List
_gui : MonitorAgentGUI
AGENT_TYPE : String
TIMEOUT : long
MonitorAgent()
shutdown()
activate()
DFSearchResults()
doneWaitTask()
startTask()
PAGTableModel:TableModel
_col_names : String
_listeners : List
getColumnCount()
getColumnClass()
getColumnName()
getRowCount()
getValueAt()
addTableModelListener()
isCellEditable()
removeTableModelListener()
setValueAt()
updated()


Template:Object
_apa : PlatformAddress
_dmi : MappingInfo
_fileCategory : String
_fileName : String
_fileSize : int
_fileSubject : String
_fileType : String
_keywords : String
_modifiedOn : Date
_smi : MappingInfo
_tid : String
getApa()
getDmi()
getFileCategory()
getFileName()
getFileSize()
getFileSubject()
getFileType()
getKeywords()
getModifiedOn()
getSmi()
getTid()
setApa()
setDmi()
setFileCategory()
setFileName()
setFileSize()
setFileSubject()
setFileType()
setKeywords()
setModifiedOn()
setSmi()
setTid()
Template()
MappingInfo:Object
_file : String
_host : String
_ip : InetAddress
_port : int
_protocol : String
url : URL
getFileName()
getHost()
getInetAddress()
getURL()
MappingInfo()
PlatformAddress:Object
_hostName : String
getHostName()
PlatformAddress()


DisplayMessage
ta : JTextArea
jf : JFrame
sp : JScrllPane
showMessage()
PrintID
index : int
dm : DisplayMessage
serviceIDNotify()
EvtConsumer
registrars : ServiceRegistrar[]
dm : DisplayMessage
disc : LookupDiscovery
templ : Template
tid : String
discovered()
doLookupWork()
doEventReg()
notify()
discarded()
ConsumerCore()
EvtGenerator
export : RemoteObject
myJM : JoinManager
dm : DisplayMessage
GROUPS : String[]
ldm : LookupDiscoveryMana...
listener : RemoteEventListener
myList : EventListenerList
templ : Template
thread : Thread
addRemoteListener()
fireRemoteEvent()
getTemplate()
startService()
EvtThread
eg : EvtGenerator
dm : DisplayMessage
run()
Template
_tid : String
_fileName : String
_fileType : String
_fileSubject : String
_keywords : String[]
setX()
getX()
EvtNotifier
myEvent : RemoteEvent
myListener : RemoteEventListener
run()
TemplateInterfaceRemote
getTemplate()
addRemoteListener()


FileTransfer
skt : Socket
os : OutputStream
is : InputStream
dm : DisplayMessage
port : Integer
FileTransfer()
sendFile()
executeFileTransfer()
transferData()
TransferInterface
getMessage()
FileDownload
host : String
dm : DisplayMessage
skt : Socket
port : int
bis : BufferedInputStream
bos : BufferedOutputStream
connect()
disconnect()
getFile()
getDataFromFile()
transferData()
executeDataCommand()
TransferInitiator
GROUPS : String[]
jm : JoinManager
localFileName : String
ldm : LookupDiscoveryManager
host : String
smi : MappingInfo
LocalFileName : String
TransferInitiator()
startService()
start()
getMessage()
MappingInfo
_file : String
_host : String
_ip : InetAddress
_port : int
_protocol : String
url : URL
MappingInfo()
getFileName()
getHost()
getInetAddress()
getURL()
FileDownloader
registrars : ServiceRegistrar[]
dm : DisplayMessage
disc : LookupDiscovery
smi : MappingInfo
dmi : MappingInfo
discovered()
doLookupWork()
getFileServer()
startDownload()
PrintID
dm : DisplayMessage
serviceIDNotify()
DisplayMessage
ta : JTextArea
jf : JFrame
sp : JScrllPane
showMessage()


LDAPAgent:FIPAOSAgent
INITCTX : String
MGR_DN : String
L_HOST : String
P_WORD : String
MY_ENTRY : String
S_BASE : String
attr : Attributes
AGENT_TYPE : String
_current_acl_encoding : String
ownID : AgentID
LDAPAgent()
main()
shutdown()
LDAPAgent()
retSmi()
shutdown()
dealWuthNA()
dealWithPA()
dealWithTA()
handleRequest()
startTask()
LServer:Object
INITCTX : String
L_HOST : String
S_BASE : String
S_FILTER : String
MY_ATTRS : String[]
MGR_DN : String
P_WORD : String
MY_ENTRY : String
ar : Attributes
jadd : JAdd
LServer()
search()
add()
del()
Document1:Object
myAttrs : Attributes
type : String
atr : Attribute
Document1()
bind()
close()
addToEnvironment()
createSubcontext()
destroySubcontext()
getAttributes()
getSchema()
list()
lookup()
search()
removeFromEnvironment()
getNameInNameSpace()
unbind()
close()
JAdd:Object
INITCTX : String
L_HOST : String
MGR_DN : String
P_WORD : String
S_BASE : String
t : Template
JAdd()
addEntry()
JDel:Object
INITCTX : String
L_HOST : String
P_WORD : String
S_BASE : String
MGR_DN : String
MY_ENTRY : String
JDel()
deleteEntry()
JSearch:Object
S_BASE : String
INITCTX : String
L_HOST : String
MY_ATTRS : String[]
ar : Attributes
tid : String
JSearch()
search()

6.2.3. CRC Cards

The CRC Cards for above-mentioned classes are given below.

Class Name: **PlatformAgent**	
Base: FIPAOSAgent	
Derived:	
Responsibilities	**Collaborators**
Receive templates	JiniAgent
Check received template for own user requirements	KnowledgeAgent
Get resource against a received template	FetchAgent
Store incoming templates	LDAPAgent

Class Name: **KnowledgeAgent**	
Base: FIPAOSAgent	
Derived:	
Responsibilities	**Collaborators**
Receive templates and reply	PlatformAgent
Compare template with user interest profiles in own knowledge base	
Receive user requirements for a resource	HomeAgent
Add user requirements as a rule in own knowledge base	

Class Name: **HomeAgent**	
Base: FIPAOSAgent	
Derived:	
Responsibilities	**Collaborators**
Create a GUI to get user requirements for a resource	
Pass user requirements to Knowledge Agent	KnowledgeAgent

Class Name: **ResourceAgent**	
Base: FIPAOSAgent	
Derived:	
Responsibilities	**Collaborators**
Create a GUI for user to specify the attributes and other information about the resource generated	
Pass the a/m info to Notification Agent	NotificationAgent

Class Name: **NotificationAgent**	
Base: FIPAOSAgent	
Derived:	
Responsibilities	**Collaborators**
Receive info about resource generated	ResourceAgent
Create the template by adding the remaining info	
Store the template	LDAPAgent
Generate notification and pass template along with it	JiniAgent

Class Name: **JiniAgent**	
Base: FIPAOSAgent	
Derived:	
Responsibilities	**Collaborators**
Receive notification along with template	JiniAgent
Pass template to Platform Agent	PlatformAgent

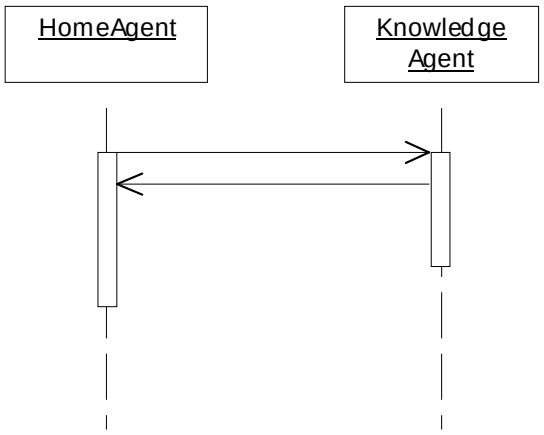
Class Name: **LDAPAgen**t	
Base: FIPAOSAgent	
Derived:	
Responsibilities	**Collaborators**
Receive template from Notification Agent	NotificationAgent
Receive template from Platform Agent	PlatformAgent
Store the template into LDAP directory	
Handle queries from Transaction Agent	TransactionAgent

Class Name: **FetchAgent**	
Base: FIPAOSAgent	
Derived:	
Responsibilities	**Collaborators**
Create a clone	PlatformAgent
Receive state from original instance	
Pass info for resource transfer	TransactionAgent
Shutdown	TransactionAgent, PlatformAgent

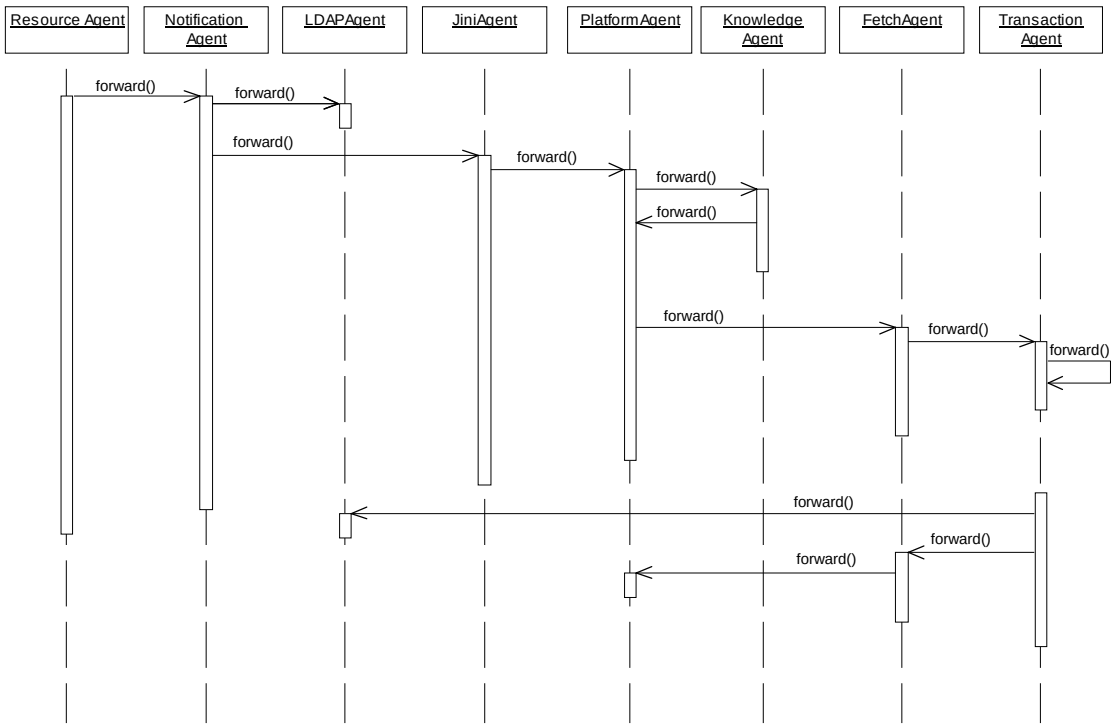
Class Name: **TransactionAgent**	
Base: FIPAOSAgent	
Derived:	
Responsibilities	**Collaborators**
Receive info for resource transfer	FetchAgent
Get mapping info against the template ID received	LDAPAgent
Open sockets	
Transfer Resource	
Generate the completion report	FetchAgent

6.2.4. Sequence Diagrams

Use Case 1 – Define User Requirements



Use Case 2 – Specify Resource Attributes



Chapter 7

7. Realization

7.1. Implementation

The implementation of the project was in two increments. In the first increment a full running prototype displaying the main functionality of the project was implemented. In the second increment the remaining functionality was added. To develop software, encompassing three different technologies, which have never been integrated like this before, some risk factor was kept in mind. The Agents require an Agent Support Platform to enable them to operate. Similarly Jini requires some http servers running on the machines in order to listen to notifications. The LDAP also has a special server, which must be installed on the network. Keeping this in view, an integration test was conducted after every major portion of implementation.

7.1.1. Development of Agents Platform

In agents after the design the next stage was to realize it. The realization process consists of the following activities:

- Ontology Creation
- Implementation of Agent Platform
- Implementation of Agents. For each agent this consists of:
    - Agent Definition
        - Task Description
        - Agent Organization
        - Agent Co-ordination

        - Utility Agent Configuration
    - Agent ‘s Task s Configuration
    - Agent Implementation

Ontology Creation. Agent understand act according to certain protocols and definitions. These protocols may be unique and specific for a certain agent platform. The definition of these protocols depends upon the type of functionality implemented on a particular agent network.

Agent Platform. The Agent Platform comprises authentication and security mechanisms. All agents coming on to a Platform must register with the local Platform AMS and DF Agents in order to operate with other agents. Implementation of these two agents is done prior to the implementation of other agents with the actual functionality.

Agents Implementation. The various functions of an agent are implemented in the form of tasks. The use of tasks functionality provided in fipa-os, enables an agent to carry out multiple tasks simultaneously. The various event-handling functions are used to enable an agent to act appropriately on different calls.

7.1.1.1. Agent Creation and Shutdown

Whenever an agent is created it registers itself with the AMS and DF so that all the other Agents come to know of its existence. After that the Agent either goes into listening mode to respond to the incoming calls or readily performs its assigned task. Once the agent is finished with the job it was supposed to do it has to shutdown. During shutting down it deregisters itself with the AMS and DF.

7.1.1.2. Agent Mobility

Agents may need to go to remote platforms for fulfilling their tasks. Three specifications for implementation of Agent mobility are Agent Invocation, Cloning and Migration. In this project Agent Cloning has been used. During execution of KMS when the Fetch Agent needs to migrate it makes a copy of itself on the remote platform and passes its state to the clone. After the clone has finished its job it passes the control back to the original Fetch Agent and shuts down.

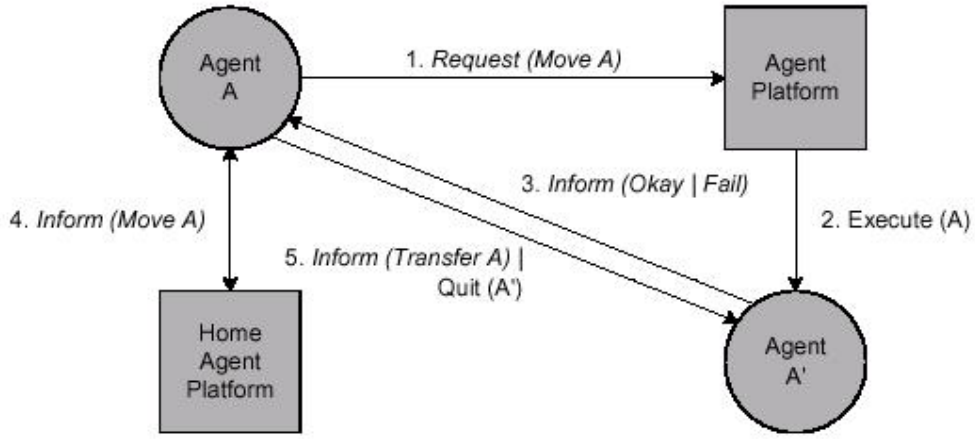


Figure 20 – Agent Cloning Protocol

7.1.1.3. Agent Intelligence

In KMS Knowledge Agent has to use intelligence while making decisions whether a resource is needed by its platform or not. For this purpose it extends Jess Agent, which implements JESS (Java Expert System Shell) API. Jess supports the development of rule-based expert systems, which can be tightly coupled to code written in the powerful, portable Java language. Jess Agent uses a Rete Object as its reasoning engine. User requirements for a particular resource are added as rules by calling functions on this Rete object.

7.1.1.4. InterAgent Communication

In this system Agents communicate and act using Task classes provided by the FIPA. Primarily there are two types of Tasks that an agent performs; hear to the incoming requests and send outgoing messages. For incoming requests all agents use their Listening Tasks and take further action based on the message they receive. For sending a message out to another Agent they use Sending Tasks where they make a message, specify the receiver and forward the message.

7.1.2. Jini Implementation

7.1.2.1. Jini Notification Service

The project required extensive Event Notifications across the network. Jini Notification Service first registers itself with the Lookup Service (LUS) and listens for the clients interested in Event Notification. Similarly clients first locate the LUS and then register themselves with Notification Service. Whenever new resource is generated the Notification service fires the Events to all registered clients. Along with notification the Template of the resource is also passed to clients.

7.1.2.2. Jini Transaction Service

When some client is found interested in particular resource its transfer is initiated by Jini Transaction service. It opens a server socket on one side and client socket on other. Client socket connects with server socket and the transfer is initiated.

7.1.3. LDAP Implementation

The directory service required for the project demanded two prime functions to be implemented. First on the list is Add and the second is search, requiring JNDI and LDAP compliant directory server to be installed. Data to be stored in the directory was to be identified and categorized. Site survey and schema design are important factors in it. After designing the schema, DIT (Directory Information Tree) was created. Databases for the directory were configured. LDAP in our project is implemented using JNDI and iPlanet Directory server. The programs are written to search, Add and delete an entry. The program to delete an entry has not been included in the project as it was not required. Complex security schemes for authentication and replication of data has been left out as they were out of the scope of the project.

7.2. Integration

The Integration of Software Agents with Jini and LDAP was a challenging task. It was done at the end of each increment. Initially the system was built as three separate modules; an Agent framework, Jini Services and LDAP directory server. For a fully running system integration of Agents with Jini and LDAP was required. As a first step the individual classes were integrated and later the entire components were made to execute together.

7.2.1. Integration with Jini

Direct integration of Jini and FIPA Agent Framework was not possible since both do not run in the same JVM. Therefore Agent abilities to communicate freely with other agents on different platforms were exploited. The Jini classes were defined as inner classes of JiniAgent, NotificationAgent and TransactionAgent classes. After getting notification and template from remote Jini Event Generator these agents pass them on to the local Agents running in FIPA using FIPA own communication channel. This enabled Jini to operate with agents as a sub layer. However the efficiency of PCs is affected due to Jini's Reggie and http servers running on client and server machines. For communication between different classes some common interfaces were implemented.

7.2.2. Integration with LDAP

Integration with LDAP was comparatively easier. The LDAP also provides well-defined API for modify, add, delete and search operations with its directory. LDAP functionality is added as part of LDAPAgent functionality.

7.3. Testing

Different types of tests were carried out to ensure that the system operates as per the desired response under all conditions. The various tests along with their motives and the outcome are appended below:

- White Box Testing
    - Ensure that all independent paths within a module have been exercised at least once.
    - Exercise all logical decisions on their true and false sides.
    - Execute all loops at their boundaries
    - Exercise internal data structures to assure their validity.
    - Amend the code as needed.
- Black Box Testing
    - Focus on the functional requirements of the software.
    - To ascertain what data rates and data volume can the system tolerate?
    - To ascertain what effect will specific combinations of input data have on system operation?
    - Amend the code as needed.
- User Testing
    - Testing by a third party (not related with the development) to get their opinion about the utility and friendliness of the application.

7.3.1. Tests and the Results

Tests.

- Passing the notification along with template to a user with some requirements, which match the template attributes.
- Passing the notification along with template to a user with some requirements, which do not match the template attributes.
- Passing a notification to two users, one with matching requirements and the other with non-matching requirements.
- Passing a notification to two users, both with matching requirements.

Test Results. The results of these tests are appended below in the same order.

- The resource was transferred successfully.
- The resource was not transferred.
- The resource was transferred to the user with matching requirements.
- The resource was transferred to both the users.

7.4. User Interfaces

The User GUI's of KMS are appended below.

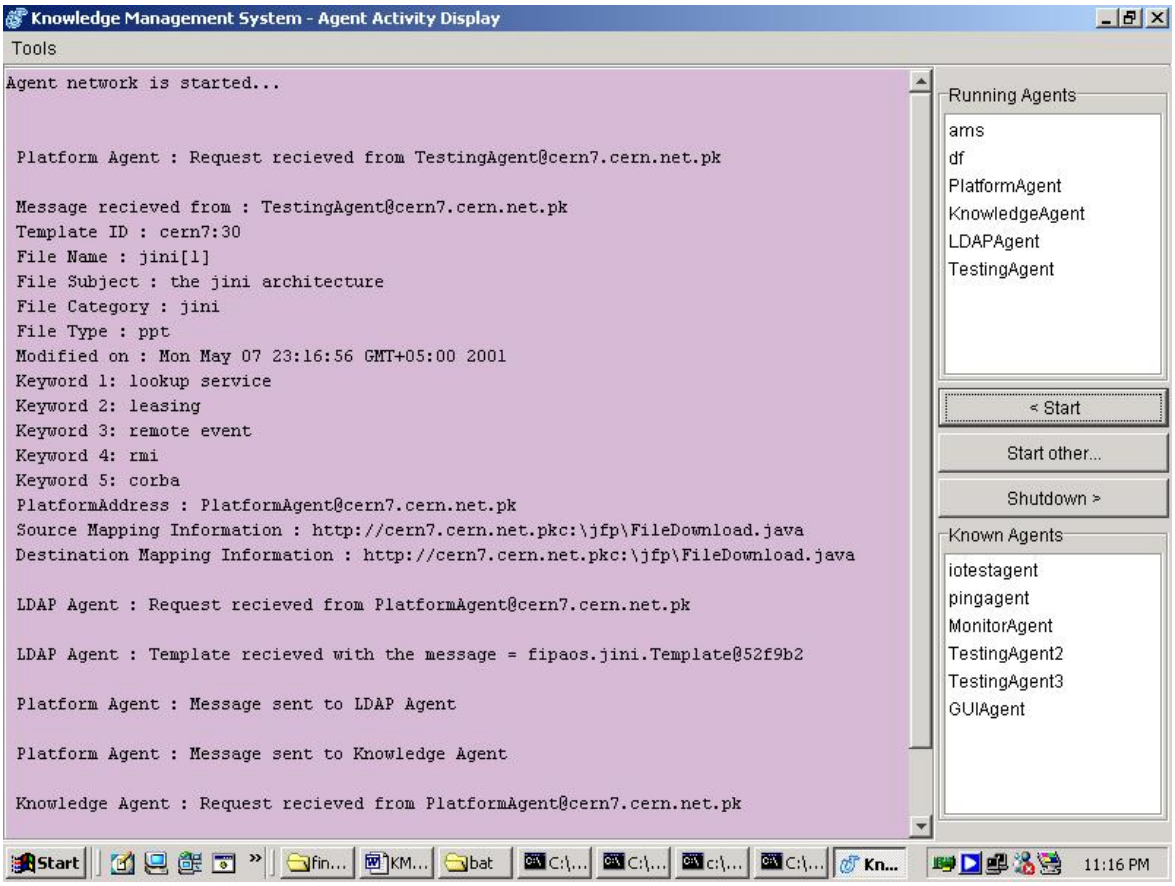


Figure 21 – Main GUI

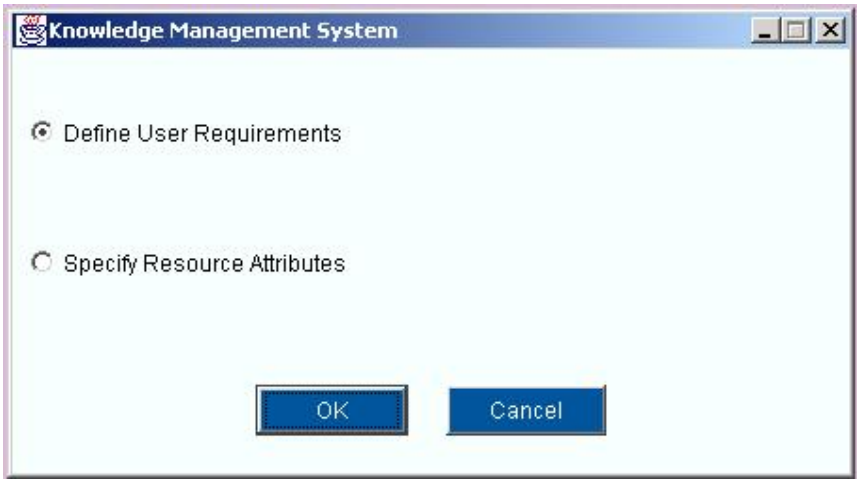


Figure 22 – User GUI

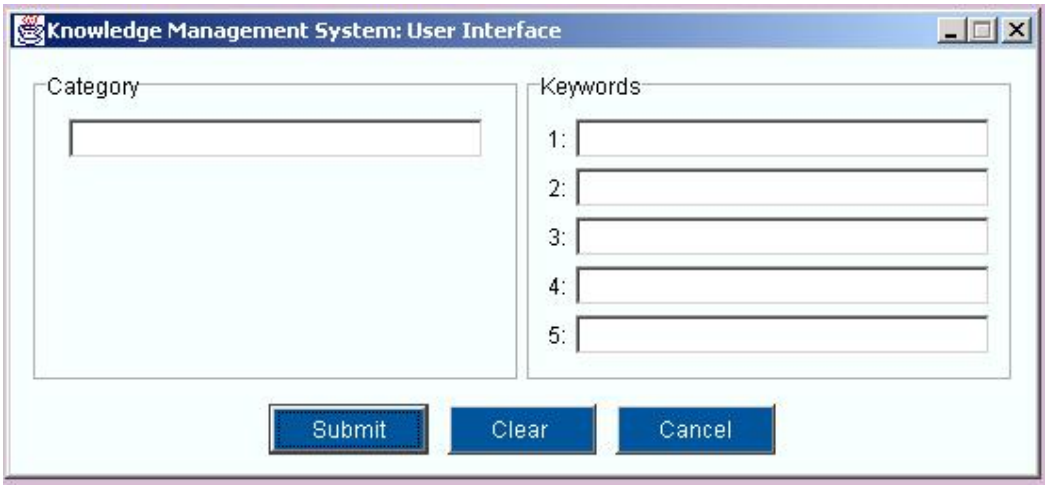


Figure 23 – Define User Requirements GUI

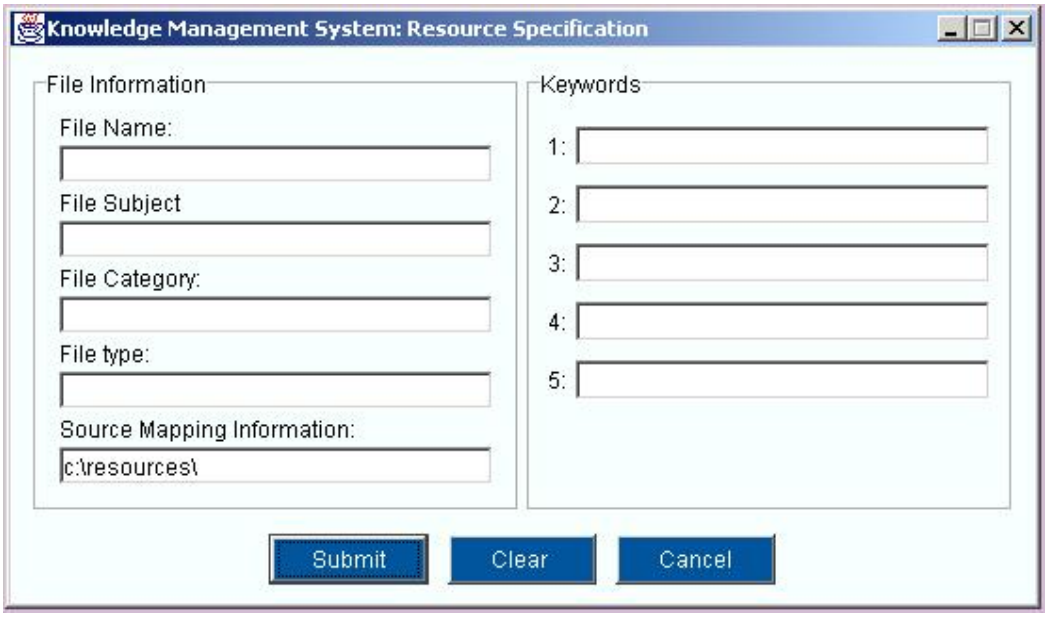


Figure 24 – Specify Resource Attributes GUI

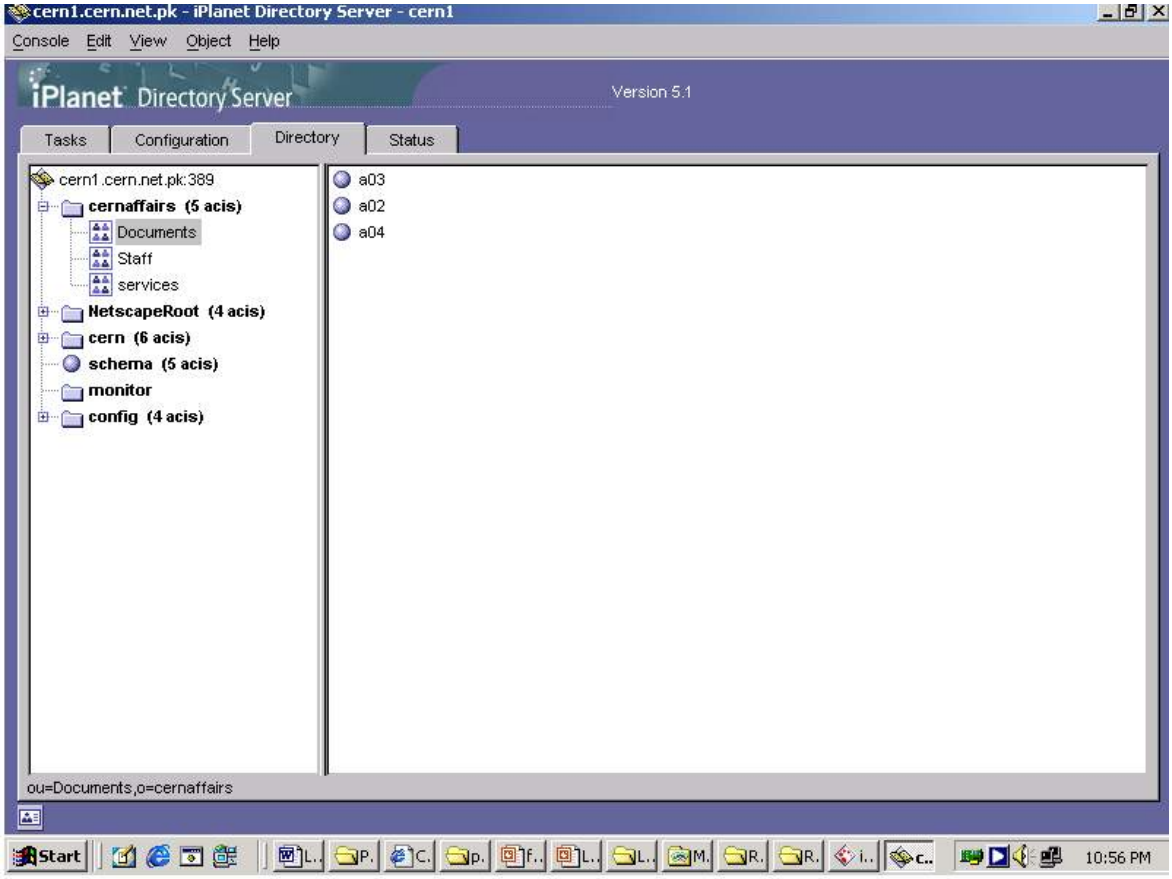


Figure 25 – LDAP Directory Server GUI

Chapter 8

8. Conclusions and Future Work

8.1. Problem Areas

This project was conceived keeping in view the Grid activities being carried out at CERN. It involved a concept, which had not been implemented before. To implement the concept, when detailed study of the requirement of the technologies was carried out, we found that we had to study the technologies of a new generation. Moreover the problems got worst when we had to integrate these technologies. Summary of the problems is given below:

8.1.1. Software Agents

- Fipa-os does not presently support agent mobility. Agent mobility is an essential feature of Fetch Agent. Three standard protocols have been defined by Fipa to implement mobility. These are Agent Migration, Agent Cloning and Agent Invocation.
- Selection of an appropriate authentication and security mechanisms is a bottleneck and yet to be explored.
- Making the Knowledge Agent intelligent is a very wide domain. Presently very simple comparisons are being carried out. However to make it more efficient more complex procedures will have to be implemented.

8.1.2. Jini

- The project required a Jini Notification Service. Unluckily Jini does not provide any standard API for Event Registration. We faced few problems implementing it before we finally succeeded.
- This project required integration of three latest technologies, which is probably not done before. We tried number of options to integrate Mobile Agents and Jini. It took our lot of time before we finally succeeded.

8.1.3. LDAP

- Directory Servers have a complex GUI and vary from vendor to vendor that is difficult to comprehend completely.
- It is very difficult to implement complex security model.
- Replication of data requires at least two LDAP servers to be either multi master or one-way replication.
- Modification of single attribute of the entry in distributed like environment is itself a complex and demanding task.

8.2. Future Work

This project is proof of a concept. It is a platform on which lot of applications can be built. Following work will make this project a wonderful contribution not only to CERN activities but also in many other Grid like environments:

- Parsing of the resource to automatically generate its template (metadata).
- Coordination of resource replication with other platforms.
- The extension of this mechanism to non-PC devices (such as cell phones, PDA etc)
- Implementation of complex template-profile matching algorithms in order to maximize relevance and focus recall.
- Integration with Globus (the Grid toolkit).
- Complex security and authentication

8.3. Conclusion

Grid Enabled Autonomous Agent Based Distributed Knowledge Management System is a research project. It has its applications in CERN's CMS Project. But that's not it; with some modifications it can be applied to any Grid enabled network. This network could be an e commerce network or even the Virtual University network. Since it is research it has lot of openings for further research in data replication on a Grid like environment. Our experience in the Project has been splendid. It was full of hard work and lots of learning. We thank All Mighty Allah for this achievement.

Bibliography

- Ian Foster and Carl Kesselman (1999). The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc. San Francisco, California.
- Thomas W. Malone, Kenneth R. Grant, and Kum-Yew Lai (1996). Agents for Information Sharing and Coordination: A History and Some Reflections. In Software Agents, ed. J. M. Bradshaw, Menlo Park, Calif.:AAAI Press.
- Pattie Maes (1997). Agents that Reduce Work and Information Overload. In Software Agents, ed. J. M. Bradshaw, Menlo Park, Calif.:AAAI Press.
- Craig A. Knoblock and Jose-Luis Ambite (1996). Agents for Information Gathering. In Software Agents, ed. J. M. Bradshaw, Menlo Park, Calif.:AAAI Press.
- James E. White (1997). Mobile Agents. In Software Agents, ed. J. M. Bradshaw, Menlo Park, Calif.:AAAI Press.
- Amit Singhal, Mandar Mitra, Chris Buckley. Learning Routing Queries in a Query Zone
- William R. Cockayne. Mobile Agents, Manning Publications, 1998.
- Jeffery M. Bradshaw. Software Agents, The MIT Press, 1997.
- Joseph P. Bigus, Jennifer Bigus, Constructing Intelligent Agents with Java, Wiley Computer Publishing, 1998.
- Bill Venners, The Architecture of Aglets, available at http://www.artima.com/underthehood/aglets.html
- FIPA FIPA 97 Specifications Part 1 & 2 Foundation for Intelligent Physical Agents Version 1.2 2000.
- http://www.enhydra.org
- FIPA-OS Developers Guide- Feb 2001 Edition.
- http://www.ldap.umich.edu
- Implementing LDAP by by Mark Wilcox.
- Understanding and Deploying LDAP Directory Services by Tim Howes, Mark Smith, and Gordon Good.

- Articles: “Getting to Know LDAP and LDAP Directories” and “Directory, Database, or Both?” by Vikas Mahajan from the site http://www.LDAPZone.com.
- http://www.internet.com ,Web developers virtual library.
- Wrox press publications found on the internet.
- ISeries LDAP: Configuring and Administering your LDAP Directory Server.
- Directory Assistance
- “A basic understanding of LDAP directory operations and working with JNDI “by John Butterfield, an associate working for Diamond Technology Partners Inc.
- Understanding LDAP and X.500, David Goodman & Colin Robbins, European Electronic Messaging Association; v2.0, August 1997.
- Inplementing Directory Services by Archie Reed.
- http://www.OpenLDAP.org
- Introduction to Directory Services and LDAP by Jeff Hodges Principal, Kings Mountain Systems.
- Lighting up LDAP: A programmer's guide to directory development, Part 2 - Choosing an LDAP server from http://www.LinuxWorld.com
- iPlanet Directory Server 5.1 Deployment Guide and iPlanet Directory Server 5.1 Administrator’s guide from http://www.iPlanet.com
- http://www.ietf.org/rfc of LDAP.
- Building Directory-Enabled Java Applications by Rosanna Lee, A JNDI tutorial.
- http://www.jini.org
- http://www.java.sun.com
- http://www.artima.com/jini (a source for JAVA and JINI developers)
- http://developer.java.sun.com/developer/products/jini
- http://www.enete.com/download/#_nuggets_
- http://www.artima.com/javaseminars/modules/Jini/CodeExamples.html
- http://www.jinivision.com/
- http://pandonia.canberra.edu.au/java/jini/tutorial/Jini.xml
- http://www.eli.sdsu.edu/courses/spring99/cs696/notes/index.html
- http://developer.jini.org/exchange/users/jmcclain/index.html
- http://sourceforge.net/projects/jini-tools

- W.Keith Edwards Tom Rodden. JINI Example by Example, The Sun Microsystems Press, 2001.
- W.Keith Edwards. Core JINI, The Sun Microsystems Press, 2001.
- John NewMarch’s guide to Jini technology.