CHAPTER 3

SYSTEM ARCHITECTURE

In our architecture as shown in Figure 3.1, image sending and receiving an image through an Intelligent Agent can be done in a compressed form. Initially the image file selected by the Sending Agent (SA) from the server and splits into different groups based on the packet size after which it is sent to the distribution agent. SA has separate agent called progressive agent, which creates a query for processing the message using two different Agent Directories. The Agent Directory Service and Agent Directory Entry. Agent Directory Service receives the query from the agent and decomposes the messages through the Agent directory Entry. The processed message is then sent to the File Distribution Agent. At the client side a group of agents used to receive the information. This chapter describes the overall architecture of this work. In our research work, we developed Multiagent systems for applications related to networks, Multiagent based hierarchical embedded differential image for progressive transmission using lossless compression, Design and evaluation of network-wide time synchronization using reference broadcasts in which the sender broadcast packets to mutable receivers in the network by using a sender and receiver agent.

3.1 PROGRESSIVE TRANSMISSION AND BROADCASTING

In this architecture, the image is transmitted from a server to various clients. The progressive transmission and broadcasting of images is achieved through agents. The important aspect of this architecture is that the
agents can communicate with each other. The image is transmitted in the form of message blocks. This is an abstract architecture, which deals with Message Transport, Agent Directory, Service Directory, and the communication that can be done using Agent Communication Language (ACL). The specification of Communication Language deal with language messages, message exchange interaction protocols, speech act, theory based communicative acts and content language representations.

Figure 3.1 Proposed multi agent architecture
Message transport specifies the path for the transfer of files through the distribution Agent. Every Distribution Agent is associated with a message transport, Agent Directory and Service Directory. Agent Directory stores all the messages that are transferred through ACL. Service Directory provides a coordination protocol description language for multi agent systems. File Distribution Agent identifies the concrete realization, based on the platform on which system works.

These can be communicated through Agent Communication Languages. This provides the interface between different concrete realizations. The sender agent selects a group of nodes to which the packets have to be sent and each receiver node holds receiver agents, which provide synchronization between receivers based on their clock frequency.

The decomposed messages as sent to the Distribution Agent, which verifies the following properties,

- Traffic Management
- Bandwidth Estimation
- Congestion Control

After verifying these properties, the Distribution Agent decomposes into different sub agents. Each sub agent then verifies all the properties of the distribution agent and selects the efficient agents and sets them as a channel for compressing the files. Threshold Sending Agents select n/2 Distribution Path Agent to process the entire distribution agent. The agent then send the information to the compression module which comprises of transformation and quantization. The encoded message is send to receiving side.
At the receiving side the message is decoded and decompressed through dequantization and Inverse Transformation.

The Distribution Agent then distributes the transformed messages and create different Distributed Agents which are given to the Threshold Sending Agent (TSA). TSA identifies the average size of the image and the average time for transmission. Threshold Sending Agents select n/2 Distribution Path Agent to send the entire message to Distribution Agents. The message loss is handled by Distribution Agents with the help of KQML (Knowledge Query Manipulation Language).

The block receiver receives the information at the destination from the distributed agents. The block merge agent merges the block which is not in a sequence, it can verify the missed block and search the agent directory entry for the missed block. If the transmission is lossy, agents retrieve the information by communicating using KQML.

The missed block is retransmitted from the sending agent and not from the server. Finally the Receiver Agent (RA) group the blocks as per sequence, which are then submitted to the receiver.

3.1.1 Agents in Sending Side

The image file is sent to the sending agent. The sending agent consist of a progressive agent as shown in the Figure 3.2, which splits the message in to different groups based on the packet size. Additionally the sending agent uses two directories, The Agent Directory Service and Agent Directory Entry. The message is finally sent to File distribution agent (FDA).
3.1.2 File Distribution Agent and Concrete Specifications

This section describes an abstract architecture of file distribution agent as shown in Figure 3.3. Such an architecture cannot be directly implemented, but instead forms the basis for the development of concrete architectural specifications. Such specifications describe in precise detail how to construct an agent system, including the agents and the services that they rely upon, in terms of concrete software artefacts, such as programming languages, applications programming interfaces, network protocols, operating system services, and so forth.
Figure 3.3 File distribution agent

In order for a concrete architectural specification, it must have certain properties. First, the concrete architecture must include mechanisms for agent registration and agent discovery and inter-agent message transfer. These services must be explicitly described in terms of the corresponding elements of the abstract architecture. The definition of an abstract architectural element in terms of the concrete architecture is termed a realization of that element; more generally, a concrete architecture will be said to realize all or part of an abstraction.

The designer of the concrete architecture has considerable latitude in how he or she chooses to realize the abstract elements. If the concrete architecture provides only one encoding mechanism for messages, or only one transport protocol, the realization may simplify the programmatic view of the system. Conversely, a realization may include additional options or features that require the developer to handle both abstract and platform-specific
elements. That is to say that the existence of an Abstract Architecture does not prohibit the introduction of elements useful to make a good agent system, it merely sets out the minimum required elements.

### 3.1.3 Security Measures for Agent Communication

In this architecture Knowledge Query and Manipulation Language (KQML) is used for Agent Communication. KQML is a message protocol designed to enable software agents to communicate with each other. In this thesis, we propose a security architecture for KQML in which symmetric or asymmetric cryptography is supported and keys are assumed to be agreed beforehand. Moreover we provide support for confidentiality, authentication, and data integrity protection. A solution using mediating agents to enable communication with crypto un-aware agents is also proposed. Another suggestion for enhancing KQML with security is proposed. Parameters for certificate management are defined leaving the format of the certificate undefined.

### 3.2 AGENT DIRECTORY SERVICES

The basic role of the agent-directory-service is to provide a location where agents register their descriptions as agent-directory-entries. Other agents can search the agent-directory-entries to find agents with which they wish to interact.

The agent-directory-entry is a key-value-tuple consisting of at least the following two key-value-pairs:
Table 3.1 Agent-directory-entry key-value-tuple

<table>
<thead>
<tr>
<th>Agent-locator</th>
<th>A globally unique name for the agent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>One or more transport-descriptions, each of which is a self-describing structure containing a transport-type, a transport-specific-address and zero or more transport-specific-properties used to communicate with the agent</td>
</tr>
</tbody>
</table>

In addition the agent-directory-entry may contain other descriptive attributes, such as the services offered by the agent, cost associated with using the agent, restrictions on using the agent, etc.

### 3.2.1 Registering an Agent

Agent A wishes to advertise itself as a provider of some service. It first binds itself to one or more transports. In some implementations it will delegate this task to the message-transport-service. As a result of these actions, the agent is addressable via one or more transports.

Having established bindings to one or more message-transport-services the agent must advertise its presence. The agent realizes this by constructing an agent-directory-entry and registering it with the agent-directory-service. The agent-directory-entry includes the agent-name, its agent-locator and optional attributes that describe the service as shown in Figure 3.4. For example, a stock service might advertise itself in abstract terms as \{agent-service, com.dowjones.stockticker\} and \{ontology, org.ontology. stockquote\}. 
3.2.2 Discovering an Agent

Agents can use the agent-directory-service to locate other agents with which to communicate. With reference to Figure 3.5, if agent B is seeking stock quotes, it may search for an agent that advertises the use of the stockquote ontology. Technically, this would involve searching for an agent-directory-entry that includes the key-value-pair \{ontology, \{com, dowjones, ontology, stockquote\}\}. If it succeeds it will retrieve the agent-directory-entry for agent A. It might also retrieve other agent-directory-entries for agents that support that ontology.
Agent B can then examine the returned agent-directory-entries to determine which agent best suits its needs. The agent-directory-entries include the agent-name, the agent-locator, which contains information related to how to communicate with the agent, and other optional attributes.

3.2.3 Service Directory Services

The basic role of the service-directory-service is to provide a consistent means by which agents and services can discover services. Operationally, the service-directory-service provides a location where services can register their service descriptions as service-directory-entries. Also, agents and services can search the service-directory-service to locate services appropriate to their needs.

The service-directory-service is analogous to but different to the agent-directory-services; the latter are oriented towards discovering agents whereas the former is oriented to discovering services. In practice also, the two kinds of directories may have radically different reifications. For example, on some systems a service-directory-service may be modelled simply as a fixed table of a small size whereas the agent-directory-service may be modelled using other distributed directory technologies.

The entries in a service-directory-service are service descriptions consisting of a tuple containing a service-name, service-type, a service-locator and a set of optional service-attributes. The service-locator is a typed structure that may be used by services and agents to access the service.

The service-directory-entry is a key-value-tuple consisting of at least the following key-value-pairs:
Table 3.2 Service-directory-entry key-value-tuple

<table>
<thead>
<tr>
<th>Service-name</th>
<th>A globally unique name for the service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service-type</td>
<td>The categorized type of the service</td>
</tr>
<tr>
<td>Service-locator</td>
<td>One of more key-value tuples containing a signature type, service signature and service address each</td>
</tr>
</tbody>
</table>

Additional service-attributes may be included that contain other descriptive properties of the service, such as the cost associated with using the service, restrictions on using the service, etc.

As a foundation for bootstrapping, each realization of the service-directory-service will provide agents with a service-root, which will take the form of a set of service-locators including at least one service-directory-service (pointing to itself).

3.2.4 Use of Directory Services for Progressive Transmission and Broadcasting

If the transmission of message block is lost, it is retransmitted from sending agent, where we have two different Agent Directories. One is Agent Directory Service and the other is Agent Directory Entry. With the help of these directories the missed block is retransmitted from the agents. So there is no need for clients and server to involve in the transfer.

3.3 AGENT MESSAGES

In the agent systems, agents communicate with one another, by sending messages. Three fundamental aspects of message communication
between agents are the message structure, message representation and message transport.

### 3.3.1 Message Structure

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

Messages can recursively contain other messages as shown Figure 3.6.

![Figure 3.6 A message](image-url)
3.3.2 Message Transport

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

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

Figure 3.7 A message becomes a transport-message
In the above Figure 3.7, a message is encoded into a payload suitable for transport over the selected message-transport. It should be noted that payload adds nothing to the message, but only encodes it into another representation. An appropriate envelope is created that has sender and receiver information that uses the transport-description data appropriate to the transport selected. There may be additional envelope data also included. The combination of the payload and envelope is termed as a transport-message.

3.3.3 Use of messages for progressive transmission and broadcasting

In progressive transmission and broadcasting the basic unit of transmission is a message block, because the agents split the image into message blocks each message consist of message transport, agent directory and service directory.

So the agent messages and message transport are useful for progressive transmission and broadcasting

3.4 MESSAGES FOR AGENT INTERACTION

In this systems agents are intended to communicate with one another. Hence, here are some of the basic notions about agents and their communications: Each agent has an agent-name. This agent-name is unique and unchangeable. Each agent also has one or more transport-descriptions, which are used by other agents to send a transport-message. Each transport-description correlates to a particular form of message transport, such as Simple Mail Transfer Protocol (SMTP), or Hyper Text Transfer Protocol (HTTP). A transport is a mechanism for transferring messages. A transport-message is a message that sent from one agent to another in a
format (or encoding) that is appropriate to the transport being used. A set of transport-descriptions can be held in an agent-locator.

For example, there may be an agent with the agent-name “ABC”. This agent is addressable through two different transports, HTTP and SMTP. Therefore, the agent has two transport-descriptions, which are held in the agent-locator. The transport descriptions are as follows:

Directory entry for ABC
Agent-name: ABC
Agent Locator:

<table>
<thead>
<tr>
<th>Transport-type</th>
<th>Transport-specific-address</th>
<th>Transport-specific-property</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td><a href="http://www.whiz.net/abc">http://www.whiz.net/abc</a></td>
<td>(none)</td>
</tr>
<tr>
<td>SMTP</td>
<td><a href="mailto:Abc@lowcal.whiz.net">Abc@lowcal.whiz.net</a></td>
<td>(none)</td>
</tr>
</tbody>
</table>

Agent-attributes:  
Attrib-1: yes  
Attrib-2: yellow  
Language: French, German, English  
Preferred negotiation: contract-net

Note: In this example, the agent-name is used as part of the transport-descriptions. This is just to make these examples easier to read. There is no requirement to do this.

Another agent can communicate with agent “ABC” using either transport-description, and thereby know which agent it is communicating
with. In fact, the second agent can even change transports and can continue its communication. Because the second agent knows the agent-name, it can retain any reasoning it may be doing about the other agent, without loss of continuity.

![Diagram](image)

**Figure 3.8 Communicating using any transport**
In the above Figure 3.8, Agent 1234 can communicate with Agent ABC using either an SMTP transport or an HTTP transport. In either case, if Agent 1234 is doing any reasoning about agents that it communicates with, it can use the agent-name “ABC” to record which agent it is communicating with, rather than the transport description. Thus, if it changes transports, it would still have continuity of reasoning.

Here’s what the messages on the two different transports might look like:

![Diagrams showing two transport-messages to the same agent]

**Figure 3.9 Two transport-messages to the same agent**

In the above, Figure 3.9 the transport-description is different, depending on the transport that is going to be used. Similarly, the message-
encoding of the payload may also be different. However, the agent-names remain consistent across the two message representations.

3.4.1 Messages for progressive transmission, broadcasting and security

At first the client will be initiated. Then the server will be initiated. The address of the image will be selected and the compression algorithm used must be selected. The agent sends the image to the client and also auto runs the progressive receiving agent. Then the server runs the progressive sending agent and gives the Internet Protocol (IP) address and file name to the progressive sending agent. Finally the Progressive sending Agent sends the image to the progressive receiving agent and according to the compression algorithm selected the output will be displayed. Every transmission is done with a help of agent messages.

In broadcasting the server is not responsible for sending the data. All server works are done by server agents. So server load gradually decreases. For processing, Server gives the information of client IP address and file path to the sender agents. In client interaction module, it creates thread for receiver agents. After processing, all receiver agents are returned. Sender Agent first finds the node availability in the network and selects the available node’s IP address and port number. Second, Each receiver agents notes the time taken by each receiver for receiving packets. In this case, receiver R1 gets the data quickly than R0 because of less traffic. Third, sorting is done based on arrival time and find minimum time taken by the machines. Fourth, find the machine order for sending the data based on sorting. Then send the file by dividing its size, because it sends the full packet sized data through less traffic path. For all these operations agent messages is a must.
Based on the security analysis three broad categories of necessary security services for multi-agent systems can be distinguished, namely, (1) those addressing the communication between agents, (2) services protecting agents against malicious platforms, and (3) services protecting platforms against malicious agents. Agent messages are useful in all the above security services.

### 3.4.2 Agent Communication by KQML

KQML messages are opaque to the content they carry. KQML messages do not merely communicate sentences in some language but rather communicate an attitude about the content (assertion, request, query, basic response, etc.). The language primitives are called performatives. At the agent level, the communication appears as point-to-point message passing.