Chapter 3

System Design of a Cloud Computing Marketplace

It has already been discussed what Cloud Computing is and what an agent is. The services provided by the cloud are smart and flexible to fulfill consumer’s requirement. A survey from several organization shows that one-third of the companies involved in the survey have already deployed cloud-based solutions [21]. Since the technology has excelled with a revolution in computing field and organizations are rapidly shifting their traditional client-server model to this new technology, there are varieties of services (IaaS, PaaS, SaaS) as provided by the cloud providers in order to fulfill the growing demands of the users. Number of cloud providers are also increasing rapidly. Selection of these services is purely requirement and data sensitivity dependent [55].

On the other hand, MAS is the another computing paradigm where multiple interacting agents capable of intelligent behavior interact with each other to solve a complex problem. Since MAS consists of multiple agents who exhibit several properties as autonomy, adaptability, cooperation, coordination, and above all, ability to interact with each other, they can be combined with their competencies to achieve a common goal. The efficient cooperation and coordination between agents make MAS more obvious choice to be used in exploring the power and improving the performance and functionality of cloud computing. Taila [86] has already explored the role of MAS in cloud computing and has named it as agent based cloud computing.

3.1 Problem Formulation

Once the consumer is convinced about the benefits of cloud computing and decides to move on to the cloud, the next step would be to choose a service provider meeting the requirements. To get the benefits of the *-as-a-services, the selection process is based on a finite set of properties and features (e.g. CPU type, memory type etc.) that may be provided by multiple providers. Existing approaches include manual searching of the providers and negotiation with them for the
services that are most suitable for them. Further, the cloud providers do not use any standardized naming convention for the services which complicate the problem. For example, compute service provided by Amazon is known as EC2 Compute Unit and the same service is referred to as Cloud Server by GoGrid and the same is known as compute cloud service by Oracle. So, it becomes a difficult task to manually obtain the list of providers and also to negotiate with those suitable providers.

Negotiation is a process of discussing something with someone (entity) in order to reach an agreement with them. Here, the entity may be two or more than two. All of us, in some way or other, knowingly or unknowingly are involved in negotiation right from our childhood. The need for negotiation with the rapid growth of the number of transactions conducted through electronic channels such as Internet has been discussed by Lau [48]. Two entities, a buyer and a seller, are involved in the process of negotiation, where both exhibit different attributes and behaviours and both are skilled with the art of bargains in their own ways. While, the seller tries to sell the product such that the profit is maximum, the buyer is satisfied if he/she is able to pay less i.e. even a single penny in the process.

Although, some popular search engines can help users to get the details of the provider’s web sites that describes the services provided by them, they are not designed to provide the list of providers providing those services and the negotiation process. The tremendous success of cloud computing shows that e-negotiation [88] will become the core of cloud computing. The e-negotiation system is a piece of software that can provide negotiation service to the user electronically.

Conventionally, e-negotiation is a process borne out by humans on certain web pages and then placing bids to offers and receiving the counter-offers by some other participants. Even though, this is a new-wave in the field, this again is a human-centered way involving knowledge and experience of human mind. The automated negotiation can be realized using software agents. So, the next step is to develop an automated e-negotiation system that may be achieved through software agents (Multi Agents System). Such flexible, autonomous MAS can be a suitable tool for service discovery and price negotiation in cloud computing. This may be termed as agent
based cloud computing where intelligent agents are designed and developed to make cloud computing technology smarter and efficient for both users and providers. Also, autonomy may help the users and providers to reach the goal and can be proved as a promising approach in the field. Deploying such kind of automated negotiation system is a trend in the future.

Therefore, to harness the full potential of cloud computing, a simple web service or application may not be sufficient and there might be a need of more adaptive, independent and above all an autonomous system which restricts the human intervention just to adjust the parameters and other preferences. So, our goal is to design, develop and implement a system which is the perfect candidate for such kind of tasks. Here the term agent based implies that agents are key abstraction and has been conceptualized in terms of agents. At this stage, our focus is on implementing a set of lightweight agents that are capable of filtering the list of providers meeting with the requirement of the cloud users and are able to negotiate with them in terms of price. The idea is to design a negotiation system in terms of software that can be used to provide negotiation service to the users and has been presented by [Srivastava, Srivastav, Bhargava] [77]. This has given us an effective agent-based solutions for cloud computing. Also, web services are the most appropriate way to deploy e-negotiation system and have been discussed by [Cao, Chi, Liu] [17]. Some researchers have believed that negotiation agents can reduce human negotiation time and are capable of identifying optimal or near optimal solution to a problem.

So, it is concluded that the convergence of the potentials of MAS and cloud computing system can result in a new system and lead to the generation of a new software which can exploit the benefits of distributed infrastructure of cloud computing in more intelligent, dynamic, flexible and automatic behavior. In such systems, agents can work together to resolve conflicts and to reach to a consensus in order to solve a problem. In implementing agent-based cloud computing, the key problems include the agents which are necessary, mode of communication, strategy required and how such system can be implemented. Till date, agents have been used in cloud computing with a focus on traffic management and service negotiation etc.
3.2 Research Design and Methodology

Research has been considered as an activity that accommodates the phenomenon of understanding. The act of research activity performed for the generation of understanding is always assisted by some research techniques or some research methods. Studies have shown that simulation plays a significant role as a methodology as far as the development of theory with few constructs and modest empirical grounding is concern. This method does not make any assumptions over other research methods while answering certain questions and has proved to be helpful for complex problems as it does not make observations by looking into the past, but rely on the future by moving forward.

The purpose of using simulation as a research method is to predict the result that can be inferred in a real situation, as this method takes a model as input governed by some rules to produce the output. Moreover, simulation makes the discovery of phenomenon possible. Also, this approach takes into account the virtual data and overcome the problem of data availability.

A jade based artefact (simulator) has been designed and developed in this study and has been named as Cloud Computing Marketplace (CCM). The requirement of such artefact in the case of this study is due to the unavailability of an integrated platform that provides the service discovery along with the dynamic e-negotiation. The assessment of the artefact is done by creating different number of selling agents. The time taken for the e-negotiation for different numbers of selling agents using the CCM will form the basis for the evaluation.

3.3 Required Language for Negotiation

The information included to start the negotiation is determined by the negotiating language. The bundle of information provided by the cloud user has a valuation (range of price including maximum and best price) and type of service along with the keyword that may identify the type of service clearly. For each such information, the user has to specify some other information
including the currency type, timeslot i.e. the deadline given to complete the task and the pay-as-you-go model type.

The information provided by the cloud seller is already stored into the database at the time of registration of the cloud provider. The providers also specify the information about the services they offer. The information provided by the providers includes their name, the type of service along with the keyword. The specific name of the service provided by them is also provided by providers. Apart, model type, valuation (range of the price from provider’s perspective i.e. minimum and best price) and currency type is also provided by the cloud provider.

Formulation: The formal description of negotiation language can be described as following:

\[ L = <A, T, V, m_i, S> \]

Where,

\( A = \bigcup A_i \), is the set of agents created to perform the task, \( A_i = \{\text{buyer agents, seller agents}\} \)

\( T: \) Deadline or timelimit, \( T = < T_{mn}, T_{mx}> \),

\( V: \) Valuation specified by cloud user and cloud providers, \( V = \{v_u, v_p\} \)

\( m_i: \) Number of agents in \( t \) where \( t \) is the negotiation round.

\( S: \) Negotiation strategy used by the agents during negotiation.

### 3.4 Participating Agents of CCM

The model, CCM, simulates the participating agents as cloud users and cloud providers. This section discusses the agents created for the experiment of the model. The main focus is on the two types of agents: Buyer Agents and Seller Agents.

A *client agent* is created for each cloud user that is using the system with the given strategic directions. This agent is responsible for creating agent in “buyer role” proactively seeking out the
potential cloud providers present in the system to negotiate with them on cloud user’s behalf. This client agent is called as a *Buyer Agent* hereafter. On the other side, provider agents are created corresponding to each provider registered in the system who are responsible for negotiating with the *buyer agent* in a “seller role”. These agents are called as *Seller Agents* hereafter.

The goal of each agent created in the system is to successfully complete an acceptable deal on its user’s behalf subject to the constraints as specified by the cloud users. These constraints could be price range (minimum and maximum in both the cases), time duration as a threshold by which the transaction has to be completed and the choice of cloud providers with which the negotiation has to be initiated with the choice of how the price over time has to be changed. *Buyer Agent* is responsible for price negotiation with *Seller Agents*. Result of the price negotiation is sent to the *Buyer Agent* who is responsible for accepting or rejecting in the negotiation. If there is more than one *Buyer Agent* participating in the marketplace, they have to compete in the auction. As soon as a *Buyer Agent* is created in the environment to act on behalf of cloud user that attempts to buy a particular service, both the *Buyer and Seller Agents* get themselves registered with the CCM agent to be able to activate in the environment and start the process. Price negotiation is the basic operation carried out in the system. *Buyer Agent* is responsible for the price negotiation with the *Seller Agents*. 
3.5 The System Architecture

In this work, the aim is to implement a multi agent environment for agent-based solution to automate cloud computing. The design of MAS architecture integrates the two technologies and the automation technique may help cloud users and cloud providers to interact with each other in a more efficient way and to reach the goal. The work has been done towards all the three different *-as-a-Service delivery classes.

Typically, the environment will act as a distributed marketplace that host cloud services and allow e-user to visit the environment and search, negotiate and purchase the required service. The user has the option to negotiate with all the providers or to choose the provider from which to make a purchase. At this stage, the assumption is that best negotiated price is the only factor determining the purchase and only providers are allowed to advertise their services after being registered. Other factors could be the best price, the safest offer, the most trusted offer etc. Cloud providers and users are created through a GUI interface that links them to their personal agents. The system is capable of creating and destroying agents dynamically.

There is only one Cloud Computing Marketplace (CCM) agent in the system which is responsible for storing, managing and providing information about the participants (Cloud Users and Cloud Providers). To participate in the marketplace all the cloud agents must register with CCM agent. The information about each cloud provider is stored in Provider Information Database (PIDB). Fig 3.2 shows the top level conceptual architecture of the implemented system, types of agents and their interaction in the system.
The architecture of the system is dynamic and as given in fig 3.3. It has mainly two layers: application layer and Internet layer [10].

- **Application Layer**: This layer consists of only one component known as standard interface. This layer allows the client to send query to the CCM. The objective of this layer is to get the requirement from the cloud user to meet his/her needs to the best. The standard interface is responsible for the interaction with the user to achieve specific task. The result is then sent to the second layer, Internet layer.

- **Internet Layer**: This layer acts as a server and contains all the components required to execute the process. This layer is skilled with the ability to accept the request, process and return the result back to the application layer. It contains the database that manages all the information of the registered cloud providers. The other components of this layer are:

1. **Agent Generation Service Manager** which is responsible for generating the necessary agents for cloud buyer and cloud providers responsible for the negotiation process,

2. **Agent Management Service** responsible for managing all the agents generated in agent generation service, retrieval of data which is responsible for searching the database for the matching needs of the cloud user received from the application layer, and;
3. **Optimization and Decision Service Manager which** performs the negotiation mechanism in order to achieve the task as specified by the cloud user.

### 3.6 E-Marketplace Scenario

A session starts with a client (cloud buyer) creating client agent through the GUI. A client agent gets all the requirements of the cloud user i.e. the type of service, key word and the range of the price i.e. the price at which the client will start to pay and the maximum price, it agrees to pay for the service and generate a *Buyer Agent*. The range of the price for *Seller Agents* is the price the provider will start with and the minimum price at which it agrees to sell a particular service. *Seller Agents* are created as soon as the cloud user enters its requirement and *Buyer Agents* are generated. Thereafter, the *Seller Agents* are engaged in negotiation with *Buyer Agent*.

![Architecture of the simulated model](image)

Fig. 3.3 Architecture of the simulated model

Following steps are involved in the e-cloud marketplace i.e. from a request made by the customer and the request is completed.

1. Both *Buyer* and *Seller Agents* will interact with the users through GUI.
2. A client agent is registered to the CCM agent and obtains an ID to be identified as soon as a cloud user initiates the query through GUI. The client agent will migrate to the site and create *Buyer Agents* and the *Seller Agents* meeting the requirement of the cloud user.
3. Both *Buyer* and *Seller Agents* implement appropriate strategies to carry out negotiation to achieve the best result for the users they represent.

4. The *Buyer Agent* gets periodically the price from all the known *Seller Agents* to provide an offer. Depending upon the offered price and on the range of price specified by its user, the *Buyer Agent* can ask the *Seller Agent* that provided the best offer to sell the service. At each negotiation the acceptable price is increased linearly, quadratic or cubically by the *Buyer Agent* and decreased by the *Seller Agents* until the maximum or minimum acceptable price by *Buyer Agent* or *Seller Agent* respectively is reached or till the deadline is expired.

5. The decision of purchase and from where to purchase is solely made by the *Buyer Agent* depending upon the preeminent offers made by the *Seller Agents*. The *Buyer Agent* will receive the negotiated price from the *Seller Agents*, compares the price and finds the best negotiated price, if any, and will make an attempt to purchase the service by informing the *Seller Agent* and the client agent to inform to the cloud user about the details: request it made, success or failure of the action, name of the provider and the negotiated price, in case of success. There could be many strategies for the offers like the best offer, the safest offer or the most trusted offers, but currently the simplest strategy of choosing best negotiated price is considered and rest of the things are left that may be included in future.

### 3.7 Ontology Defined for the CCM

Ontology is itself a subject of research in the field of Artificial Intelligence (AI). Originally, the term ontology adopted from the branch of philosophy called metaphysics, can be defined in many ways. It may be considered as a formal description of knowledge for a particular domain. There is a distinction between the terms as used in philosophy and computer science [30]. Whereas the philosophical view of the term considers the categorical analysis of the entities, computer science preview this term technically to create engineering models of reality or computational artifacts as used by software models.
Gruber [29] was the first one to define the term as “an explicit specification of conceptualization” in contrast to the definition given in philosophy which is concerned with the study of what exists. The notion “conceptualization” was initially defined by Genesereth and Nilsson [27] as “the object, concepts and other entities that are presumed to exist in some area of interest and the relationships that hold them”. This area of interest now typically called as domain. The term conceptualization is a simple and abstract view of something that is being represented for some purpose. Borst [13] later in 1997 added two terms to the definition stating ontology as a “formal specification of a shared conceptualization” where conceptualization was expressed as a shared view between different parties and should be expressed formally in a machine readable format. In 1998, Studer et.al. [79] combined the two concepts and produced a new definition as “An ontology is a formal, explicit specification of a shared conceptualization”.

The ontology engineers consider related entities and codify them as concepts and relationships. In computer science terminology, this codification refers to defining a set of concepts, characteristics and relationships for a particular domain. Ontology acts as a structural framework used in fields such as artificial intelligence or information architecture etc. to organize information and concepts. The set of concepts, characteristics and relationships are called vocabulary which is used by the researchers to share common information in a particular domain and helps the machine to interpret the concepts and their relationship in an organized way. For a particular domain, ontology is not just a representation but an agreement of knowledge in that domain.

Since ontology is an agreement of knowledge (representation) over a domain, its development can be considered as a cooperative process involving different people at different locations. Defining ontology for a domain is analog to defining an array of records with fields name in programming language or a table with attributes name for a database application. It provides the information of the data structures of an application. Fig. 3.4 depicts the Ontology and its Constituents.
Taking into account the problem statement discussed above, the ontology, addressed with respect to the domain used includes five concepts, one predicate and one agent action.

The concepts designed are as follows:

- **Cloud_Service**: has four slots, namely, `cs_serviceid`, `cs_model`, `cs_currency`, `cs_keyword`. All the slots have single cardinality. This means that every instance of Cloud_Service needs exactly one ID, one model, one currency value, and one keyword.

- **Cloud_Provider**: This concept has three slots. These are named as `cp_providerid`, `cp_bestprice` and `cp_minprice`. The `cp_serviceid` slot denotes the ID of the cloud provider registered, whereas the slots `cp_bestprice` and `cp_minprice` denotes the best price (or initial price) the provider has initially quoted to sell the service and the least price, the provider will negotiate with the cloud user and agrees to sell the service respectively.

- **Cloud_Buyer**: This concept consists of `cb_bestprice` and `cb_maxprice` slots. Like provider’s best price and minimum price, cloud buyer also quotes the initial price to pay for a service and the highest price, the buyer pay for a service after negotiation with the provider.
• **Cloud_TimeLimit:** The slots of this content include the time limit, in terms of hours, minutes and seconds, the cloud user is ready to negotiate with the providers for a particular service. The slots are defined as \texttt{TL\_HRS}, \texttt{TL\_MINS}, and \texttt{TL\_SECS}.

• **Cloud_Result:** This concept is responsible to hold the details of the result obtained after the final agreement between cloud provider and cloud buyer. The slots used in this concept are \texttt{CR\_TITLE} and \texttt{CR\_PROVIDERNAME} along with a super-schema \texttt{CLOUD\_SERVICE} so as to inherit all the slots included in its super schema along with its own.

In order to have a negotiation of agents on cloud services in a CCM settings, one Action type is modeled: \texttt{SELL}. This agent action is applied to a service and is used to negotiate on the price of services. Apart from concepts and agent action, one predicate \texttt{RESULT} is used to associate the negotiated price (final price) with the cloud service searched for.

### 3.8 Technical Background

In MAS, the most and ever relied upon technique for coordinating agents is *Negotiation*. It consists of repeated contract of agents on some assignment involving exchange of some task (proposed price) based on bidding. A group of agents is involved in the communication process of negotiation so as to reach a mutually accepted agreement on some matter (Bussmann and Muller, 1992). There can be two types of negotiation depending on the behavior of the agents involved: Competitive and Cooperative. Competitive negotiation, as the name suggests, is involved when agents have independent goals. And, Cooperative negotiation is involved when agents have a common objective to accomplish a task.

In an automated negotiation system, negotiation is a social interaction between agents and many agent oriented software development platforms are available with the required infrastructure of MAS which works on standard agent technologies and provides the development environment to build agent system. When agents are involved in negotiation, the points of concern are negotiation protocol, issues and mechanisms used for negotiation. The specification that is
considered here is The Foundation for Physical Agents (FIPA) [9] and the software framework that has been used to write agent application is JADE (Java Agent DEvelopment framework) which is FIPA compliant and provides a contract net protocol in general for such kind of issues. The other concern to use JADE is the support for Agent Communication Language (ACL) defined in later sections which allows agents to exchange information and services with each other in order to complete a task. The other important issue is Ontology definition supported by JADE which is defined as Java objects and which makes the exchange and parsing of messages between agents easy.

### 3.8.1 FIPA (The Foundation of Intelligent Physical Agents)

*FIPA* is the acronym of *The Foundation for Intelligent Physical Agents*. It is a Swiss non-profit organization that was established in 1996 to produce standards relating to generic software agent technology (MAS System). The aim of developing FIPA was to define a set of standards for both; implementing MAS where agents can be executed and to specify how agents should communicate and interact themselves in MAS (agent platform). The key feature of FIPA is to promote inter-operable agent applications and agent systems and to standardize the protocols for external interaction (behavior) and platform services for agent-related ideas. Some of the protocols have been successfully promoted to standard status while some have been developed but remain incomplete. Of these, the most important for the MAS are agent communication, agent management and agent architecture. Further the key concept associated with each of them is also discussed.

#### 3.8.1.1 Agent Communication

One of the primary characteristic of MAS is communication as agents have to communicate with each other, with the users and with system resources in order to cooperate, coordinate or negotiate. Consider, for example, one person saying something to another person. This communication needs a language as a medium. Also, communication here involves

1. Two humans: a speaker (sender) and a listener (receiver),
2. Two human processes: a sender expressing something (e.g., some message) and a receiver interpreting the message.

Similarly, when agents interact with each other, they need a medium to communicate or a special language to communicate. This language, in computer science, is known as Agent Communication Language (ACL), and is based on speech act theory (Searle, 1969) [72]. ACL provides a contour between the communicative acts and content language. The first ACL was KQML (The Knowledge Query and Manipulation Language), developed in 1990’s by US government’s ARPA Knowledge Sharing Effort. Currently, the most used and studied ACL is FIPA-ACL. According to FIPA-ACL, messages represent actions or communicative acts (CA). These acts are known as speech acts or performatives. There are set of 22 communicative acts in FIPA-ACL based on ARCOL (Artemis Communication language) proposal of France Telecom [36]. Few of the most commonly used acts are inform, request, agree, not understood and refuse. These are described later.

Based on these CAs, FIPA has refereed some Interaction Protocol [7] for Interaction Process (IP) or IP protocol, which allows CAs to be used in the form of message sequence or workflows to support multi message actions, including information exchange and task delegation. There are some FIPA standardized IPs such as request, recruit, subscribe, auction and contract net. These have been discussed later.

3.8.1.2 Agent Management

Apart from agent communication, another important work related to agent addressed by FIPA is agent management - a normative framework within which FIPA-compliant agents exists, operates and are managed. It represents a logical reference model for the creation, registration, location, communication, migration and operation of agents. The components of this model consist of:

- Agent Platform (AP): This is the physical environment, where the agents are deployed. The AP includes machines, operating systems, FIPA agent management components, the agents and any other additional software support. The internal design of AP is not subject to FIPA specification, rather it is purely developer dependent of the system. An AP may
be extended across many computers and the agents of this AP do not have to be located on the same host.

- **Agent**: This is a computational process that offers one or more computational services and an inhabitant of AP. The only issue FIPA addresses about agent is the structure and encoding of the information in the form of messages to be exchanged in MAS and is not concerned with the design of these agents. An agent must belong to at least one AP, must have a unique identity to be identified unambiguously as described by FIPA Agent Identifier (AID) and must be registered at some number of transport address so that it can be contacted.

- **Directory Facilitator (DF)**: This is an optional component and acts as yellow pages services for other agents in AP. It updates itself by keeping the most current information about agents, with the list of agents and their capabilities (services), registered with them.

- **Agent Management System (AMS)**: This is the mandate component of an AP. The responsibility of this component is to manage the operations (creating, deleting and migration of agents) in AP. It provides the naming service to ensure that each agent in the platform is identify unambiguously. For this, each agent must be registered with AMS to obtain a unique identity. It is also responsible for shutting down the whole platform.

- **Message Transport Service (MTS)**: This provides a mechanism to transport FIPA-ACL messages between agents that may be local to a single AP or between different APs. It is used to describe a message envelope with a set of mandate parameters detailing to, from, date and acl-representation to some optional parameters.

Fig.3.5 describes the components of agent management reference model.
3.8.1.3 Agent Architecture

The agent architecture deals with the abstract entities which are required to build agent services and an agent environment. Built according to the Abstract Architecture, agent systems can interoperate with each other through transport gateways including logical (from one representation to another) and transport transforms. It permits the creation of multiple concrete realizations. It mandates the use of ACL message structure, message transport, agent directory service and service directory. Other aspects are optional. For example, if one agent system transmits ACL messages using the OMG IIOP (Object Management Group Internet-Inter ORB Protocol) protocol and another agent has used IBM’s 417 MQ-series enterprise messaging system, then architectural abstraction helps in arriving at senders and receivers identification along with encoded message and transportation. The overall goal of agent architecture is to create systems that seamlessly integrate within a specific environment while interoperating with agent systems residing in separate environments.

3.8.2 Key FIPA Specifications

Although there are 25 specifications which have been standardized till date, only few specifications have been discussed in this section. The complete list has been annexed at the end.
3.8.2.1 FIPA Abstract Architecture Specification

As discussed previously, the primary focus of FIPA Abstract Architecture is to create semantically meaningful message exchange between agents which may be using different messaging transports, different Agent Communication Languages, or different content languages. It describes how two agents are located and communicate with each other by registering themselves and exchanging messages at an abstract level. As mentioned earlier, agents can communicate with each other through messages which are exchanged between them and which are based on speech acts. These messages are encoded in agent communication language. Agents are provided with some support services through services including standard services of agent directory services, message transport services and service directory services apart from others.

Some of the most important items specified in FIPA Abstract Architecture are:

Agent Messages: The key idea of communication between agents is message. There are three fundamental aspects of messages to be considered are: message structure, message representation and message transportation. The structure of the message is key-value-tuple written in FIPA ACL. The message content is represented in a content language such as FIPA-KIF or FIPA-SL. The most commonly used is SL which has been discussed later. The messages can be inline; i.e one message can contain another message as content. Apart from this, some other parameters such as sender and receiver’s name are also contained in the message. Sender and receivers are identified with a unique name.

Message Transport Service: Transportation of message between two agents, sender and receiver is identified with this service and is considered to be mandatory field in FIPA agent system.

Service Directory Service: This service provides a collectively shared repository through which agents can discover services. The types of services are message transport services, agent directory services, gateway services and message buffering. This field is also considered mandatory element for FIPA agent system.
Agent Directory Entry: The key role of this service is to provide the location of the agent where it has been registered and to search for the agent with which they want to interact. The description of these agents are published in agent directory entries. This is also a mandate field for agent system.

3.8.2.2 FIPA-ACL Message Structure Specification

As communication between agents are done through message exchanging, there is a specification defined for FIPA-ACL message parameters which ensures the interoperability with a standard set of ACL message structure and provides a well-defined process for maintaining this set. A FIPA-ACL message consists of one or more parameters. Although, need of parameters for effective agent communication vary with situation, the only parameter required by all ACL messages is performative. Other parameters that ACL messages contain are sender, receiver and content parameter which are optional. Table 3.1 shows the list of ACL Message parameters.

Agents may reply a not-understood message, if unable to process or is not able to recognize parameter or its value. A user-defined parameter can also be included in the list of message parameters. A string “X-“ has to be preceded for an additional non-FIPA standard parameter.

Table 3.1 ACL message parameters [8]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performative</td>
<td>Type of the communicative act of the message</td>
</tr>
<tr>
<td>sender</td>
<td>Identity of the sender of the message</td>
</tr>
<tr>
<td>receiver</td>
<td>Identity of the intended recipients of the message</td>
</tr>
<tr>
<td>reply-to</td>
<td>Which agent to direct subsequent messages to within a conversation thread</td>
</tr>
<tr>
<td>content</td>
<td>Content of the message</td>
</tr>
<tr>
<td>language</td>
<td>Language in which the content parameter is expressed</td>
</tr>
<tr>
<td>encoding</td>
<td>Specific encoding of the message content</td>
</tr>
<tr>
<td>ontology</td>
<td>Reference to an ontology to give meaning to symbols in the message content</td>
</tr>
<tr>
<td>protocol</td>
<td>Interaction protocol used to structure a conversation</td>
</tr>
<tr>
<td>conversation-id</td>
<td>Unique identity of a conversation thread</td>
</tr>
<tr>
<td>reply-with</td>
<td>An expression to be used by a responding agent to identify the message</td>
</tr>
<tr>
<td>in-reply-to</td>
<td>Reference to an earlier action to which the message is a reply</td>
</tr>
<tr>
<td>reply-by</td>
<td>A time/date indicating by when a reply should be received</td>
</tr>
</tbody>
</table>
3.8.2.3 FIPA-ACL Communicative Act Library Specification

According to FIPA-ACL, communication between agents is defined as function or action which is called as Communicative Acts (CA) based on the act of communication. FIPA specifies a standard library of all CAs called as FIPA Communicative Act Library (CAL). Table 3.2 shows the list of all communicative acts in FIPA. The FIPA CAL specifies functions which define the actions for CAs based on speech act theory [72]. The functions include *interrogative* (for querying some information), *exercitives* (asking for the action to be performed), *Phatics* (for establishing, prolonging and interrupting the communication), *paralinguistics* (for relating one message to another) etc.

3.8.2.4 FIPA-SL Content Language Specification

The FIPA has defined a Semantic Language (SL) to specify the intentional meaning for the FIPA CAs. It has defined CAs logic of mental attitudes and actions. It has been formalized as a first order modal language. SL has been represented as a string expression grammar which in particular, has been defined to be a sub-grammar of the more general s-expression syntax. Content expressions are represented in terms of action expressions or prepositions, which in turn are represented as well-formed formulas (wff). Wff consists of terms and constants. Details of all such things have been presented in the specification documents.

FIPA has also categorized SL into three subsets: SL0, SL1, and SL2. Basic difference in all these three subsets lies with the type of operator it supports. FIPA SL1 is an extension of SL0 by adding Boolean connectives such as *not, or, and* in the propositional expressions. Similarly, SL1 is extended by adding construction, logic modal operators and the action operator for SL2. Different SLs have been used to define different CAs.
3.8.2.5 FIPA Request Interaction Protocol Specification

The FIPA Request Interaction Protocol (IP) allows the agents to send request to other agents to perform some action. The agent generating the request is known as Initiator and the other one is known as Participant. The sequence flow of the protocol has been depicted in fig 3.6. The value of the protocol parameter of the ACL Message is fipa-request to identify this protocol.
The flow of the protocol is: the Initiator initiates the request and the Participant process the request and makes decision whether to accept or refuse the request. If a refuse decision is made, then the Refused is set to true and refused message is communicated to the Initiator, otherwise agreed is set to true. Once the request has been agreed upon, the Participant has to communicate either of the following:

- A failure in case it fails to fulfill the request,
- An inform-done, if it successfully completes the request or
- An inform-result, if in case it wishes to inform both completion of request and the result to the Initiator.

![Fig. 3.6 The FIPA Request Interaction Protocol [8]](image-url)
A unique, not null identity, conversation-id parameter is assigned to all the interaction using this protocol by the Initiator to uniquely identify the conversation which helps agents to manage its communication strategies and other activities. A not-understood message can be communicated by Participant if it is not able to understand the communicated message in IP at any point. This action may terminate the entire IP or may nullify or void the commitments made during the interaction. At any point, the Initiator may also cancel the interaction protocol using meta-protocol. The structure of this protocol is shown in fig 3.7.

![Fig.3.7 The FIPA Cancel Meta-Protocol](image)

### 3.8.2.6 FIPA Contract Net Interaction Protocol Specification

This is a more complex Interaction Protocol. In addition to generate the request by one agent to other agents (one or more) in order to perform actions, this IP also optimizes a function based on some character. The character may be earliest time completion or fair distribution of task. But most commonly it is used to optimize a character defined in terms of price. In response to a request generated by an Initiator agent, any number of participants can respond with a proposal and rest of them may refuse. A request for proposal to perform some action by Initiator agent is issued by setting the performative parameter to `call for proposals (cfp)` CA. The cfp CA
specifies the task and the conditions; Initiator is imposing in order to complete the task. Upon receiving the cfp, Participants generate \( n \) responses, of which \( j \) are acceptable for the task to perform while other \( n-j \) are refusal. The propose CA is used to specify acceptance of the Participant and refuse is used to specify the refusal to perform the task stating the condition for refusal. The propose CA includes the preconditions setting out for the task as specified above. The cfp CA works on a deadline by which the reply should be sent by the Participant to the Initiator. This feature of cfp helps Initiator not being left waiting indefinitely for the Participant in case they are not able to reply either with a propose or a refuse act. After the deadline is completed, the proposals are automatically rejected with the reason of late proposal. The deadline is specified in the ACL message by the parameter `reply-by`. Once the deadline is passed, the Initiator evaluates the proposal and sends accept-proposal CA to the selected agents and reject-proposal to the rest of the agents. After completion of the task, Participant sends a message of completion to the Initiator. This message could be: inform-done, to inform the completion of task or inform-result, the explanatory version with the result of the task or a failure message to inform the failure of the task. The IP of this specification is depicted in fig 3.8.

### 3.8.2.7 FIPA Agent Message Transport Service Specification

FIPA Agent Message Transport Service (MTS) specification deals with message transportation of FIPA ACL message between inter-operating agents. It uses the Message Transport Protocol (MTP) to physically transfer the message. The agents may be local agents belonging to the single Agent Platform (AP) or the agents may belong to different APs. An agent message consists of two parts: a message envelop consisting of transport information and the ACL message of agent communication called message payload. Different MTP may use different representation to express a message envelop but must express same terms, semantics and action performed. In general, message envelop consists of some collection of parameters. These parameters are name/value pair. The mandatory parameters are: to, from, date and acl-representation. Other parameters are optional. The MTS is provided through an Agent Communication Channel (ACC), given any AP. In order to transport message, the ACC may require to obtain the information from AMS and DF of other AP Services or from its own AP. In order to provide the transport service, ACC can only transfer the received message according to the transport
instruction received in the message envelop. ACC can read the message envelop and it is not required to parse the message payload. While handling a message, ACC may add new information to the message envelop, but cannot overwrite existing information. Also, ACC can add its own parameters in the message envelop, however these parameters will override the existing parameters with same names.

![Fig.3.8 The FIPA Contract Net Interaction-Protocol [8]](image)

The message envelop contains some parameters, described in table 3.3, are used to determine the transportation behavior of an ACC for message delivery. The \( \text{to} \) parameter specifies the recipient of the message and is generally expressed in terms of AIDs.
There are three options available for an agent to send the message to an agent on the remote AP:

- Agent A sends the message to its local ACC using standard interface, which in turn sends the message to the ACC on the remote AP using suitable MTP and the remote ACC is responsible for delivering the message to the agent B,
- Agent A can directly send the message to the remote ACC, where Agent B resides, which in turn delivers the message to the desired agent. Agent A must be aware of and have access to one of the MTP’s interfaces of the remote ACC to use this method and
- In this case Agent A and Agent B communicate with each other through direct communication method. Sending and receiving agents are responsible for message transfer, buffering of message, addressing and error handling. FIPA does not cover this method of communication.

### Table 3.3 The FIPA message envelop parameters [8]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>to</td>
<td>The names of the primary recipients of the message</td>
</tr>
<tr>
<td>from</td>
<td>The name of the agent that sent the message</td>
</tr>
<tr>
<td>comments</td>
<td>Text comments</td>
</tr>
<tr>
<td>acl-representation</td>
<td>The syntax representation of the message payload</td>
</tr>
<tr>
<td>payload-length</td>
<td>The length in bytes of the message payload</td>
</tr>
<tr>
<td>payload-encoding</td>
<td>The language encoding of the message payload</td>
</tr>
<tr>
<td>date</td>
<td>The creation date and time of the message envelope</td>
</tr>
<tr>
<td>intended-receiver</td>
<td>The name of the agents to whom this instance of a message is to be delivered</td>
</tr>
<tr>
<td>received</td>
<td>A stamp indicating the receipt of a message by an ACC</td>
</tr>
<tr>
<td>transport-behaviour</td>
<td>The transport requirements of the message</td>
</tr>
</tbody>
</table>

### 3.9 Data Flow Diagrams of the System

Fig. 3.9 is a context level diagram, an overview of a system showing the boundaries and sources/sinks interacting with the system and the major flow of information between the system and the entities. It gives the top level view of the system.
Fig 3.10 gives the Level 0 diagram of the system. It shows all the processes comprising of the overall system at the high level of abstraction indicating the information moves from and to each process. It also shows data stores of the system.
The work flow of the automated negotiation for the cloud computing model is shown as in fig 3.11. It illustrates the different stages involved in and the relation between them, depicting the working of the system.

Fig. 3.11 Workflow of the model
3.10 The Negotiation Algorithm

The CCM adopts a style of negotiation, where a buyer agent is negotiating with each seller agent in parallel. Upon receiving a request from buyer agent, it is broadcasted to all the seller agents. Offers generated by seller agents are sent to the requesting buyer agents instead of broadcasting the offer to other seller agents. The model used in the negotiation process is Bilateral Negotiation Model.

The algorithm used in the system is given below:

1. User Agent is created upon receiving the specifications from the user.
2. The received specifications are searched in the database.
3. As many seller agents are created as the result of the match from the database.
4. Each seller agent calculates the price and sends the manipulated price to user agent.
5. User agent compares the price with that calculated by it based on user specifications and sends accepted or refused proposal to the seller agent.
6. On acceptance proposal, the buyer agent will sends the price to the user console with the detail of provider’s name.

The utility functions $U$ have been designed for both consumer and provider to calculate the acceptable price. Each time a new offer price is generated with the help of utility function $U_{\text{pro}}$ of provider, it is being sent to the consumer agent, which in turn uses its utility function $U_{\text{con}}$ to evaluate the acceptable price on behalf of consumer. CCM implements a straight forward negotiation protocol with three negotiation strategies to choose over by the cloud users: linear, quadratic and cubic functions for increasing (or decreasing) its bid for a particular service over time. These functions are presented for both, cloud users as well as cloud providers. Fig 3.12 shows the graph of all the three negotiation strategies for both buyer agent and seller agent. The graphs show the nature of the price as a function of time.
Fig 3.12 Graphs showing the three negotiation strategies for both, cloud users and cloud providers
The **Negotiation Algorithm** for accepting an offer is given as follows:

**Step 1: Seller agent’s offer price:** For calculating the offer price of the seller agent, the utility function used is $U_{pro}$ for seller agent. The $U_{pro}$ is given by

$$U_{pro}(acceptP) = \text{maximum} - \epsilon$$ (1)

Where $\epsilon$ is the minor increment in the price of the service and is calculated as

- For linear negotiation:
  $$\epsilon = \text{deltaP} \times (1 - (\text{elapsedTime} / \text{deltaT})$$
- For quadratic negotiation:
  $$\epsilon = \text{deltaP} \times (1 - (\text{elapsedTime} \times \text{elapsedTime} / \text{deltaT} \times \text{deltaT})$$
- For cubic negotiation:
  $$\epsilon = \text{deltaP} \times (1 - (\text{elapsedTime} \times \text{elapsedTime} \times \text{elapsedTime} / \text{deltaT} \times \text{deltaT} \times \text{deltaT})$$

Where, 

- $\text{deltaP}$ is the difference between the cloud providers best price and minimum price,
- $\text{elapsed Time}$ is the difference between current time and initial time, and
- $\text{deltaT}$ is the difference between time limit and initial time

And $\text{maximum P}$ is the highest range of price as prescribed by the cloud provider

**Step 2: Buyer Agent’s requested price:** To calculate the initial minimum requested price of the buyer agent a utility function $U_{con}$ has been referred and the acceptable price has been calculated as follows:

$$U_{con}(acceptP) = \text{minimumP} + \epsilon$$ (2)
Where \( \Delta \) is the minor increment in the price of the service and is calculated as

- For linear negotiation:
  \[
  \Delta = \delta P \ast (1 - (\text{elapsedTime} / \delta T)
  \]

- For quadratic negotiation:
  \[
  \Delta = \delta P \ast (1 - (\text{elapsedTime} \ast \text{elapsedTime} / \delta T \ast \delta T)
  \]

- For cubic negotiation:
  \[
  \Delta = \delta P \ast (1 - (\text{elapsedTime} \ast \text{elapsedTime} \ast \text{elapsedTime} / \delta T \ast \delta T \ast \delta T)
  \]

Where,

\( \delta P \) is the difference between the cloud providers maximum price and the best price, elapsed Time is the difference between current time and initial time, and \( \delta T \) is the difference between time limit and initial time

And minimumP is the minimum range of price as prescribed by the cloud user.

**Step3: Matching:** By comparing the prices calculated in eq. (1) and (2), the price will be decided. The condition for comparing prices is

\[
U_{\text{conf}}(\text{acceptP}) < U_{\text{pro}}(\text{acceptP})
\]

The system is engaged in a loop that can be terminated if either of the following conditions is satisfied:

- Running out of Time (\( T_{dl} \leq 0 \), where \( T_{dl} \) is the dead line in terms of time)
- Reached an agreement (Cheapest contract)