TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SANDEEP KUMAR SHRESTHA

FIPA-COMPLIANCE OF HTML5 AGENT FRAMEWORK

Master of Science thesis

Examiner: Prof. Kari Juhani Systä
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 9th September 2015.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
SANDEEP KUMAR SHRESTHA: **FIPA-Compliance of HTML5 Agent Framework**
Master of Science Thesis, 58 pages
April 2015
Major: Software Engineering
Examiner: Prof. Kari Juhani Systä
Keywords: HTML5, mobile agent, framework, FIPA, interoperability, compliance, MAS, FIPA-ACL, specifications, agent platform.

In agent-oriented architecture, systems are built on autonomous components called agents. Agents exist and operate in an agent environment/platform. When an agent environment contains two or more agents, it is called Multi-agent System (MAS). Like humans, agents have an ability to cooperate, coordinate, negotiate and interact with each other to resolve problems on the behalf of their users. Moreover, agents in agent environment can reach beyond its system environment and interact with agents in other third-party agent environments for co-operative problem solving. Agent systems developed by different developers possess architecture specific features and implementation. These differences among agent systems prevent interoperability between agents existing in different agent environment. Therefore, mechanisms that allow agents and/or MASs to interoperate are needed.

It is easier to rationalize the use of agent systems based on existing well known standards like FIPA than on self-made standards. The HTML5 Agent Framework developed in TUT has its own architecture specific features and implementation. The main purpose of this thesis is to analyze how HTML5 Agent Framework can be made FIPA compliant. An agent system that conforms to FIPA specifications is a FIPA-compliant system. A FIPA-compliant system can interoperate with other heterogeneous agent systems that are FIPA-compliant as well. The conversion of MAS into a FIPA-compliant system is one way of guaranteeing interoperability between different MASs. FIPA is a standard body that provides specifications for developers of agent systems. It promotes agent-based technologies and interoperability of its standards with other agent-based technologies that facilitate the end-to-end interworking of agent systems in modern commercial and industrial settings. In this thesis, the current implementation of HTML5 Agent Framework is mapped with FIPA standards. This thesis presents analysis to make HTML5 Agent Framework a FIPA-compliant agent system. Moreover, possible solutions to make HTML5 Agent Framework compliant to FIPA are suggested. A proof of concept is also implemented. It can establish simple communication between HTML5 agent and FIPA-compliant JADE agent.

PREFACE

This Master of Science Thesis has been written for Department of Pervasive Computing at Tampere University of Technology (TUT), Tampere, Finland.

First and foremost I would like to express my sincere gratitude to my supervisor Prof. Kari Juhani Systä for providing me continuous support, guidance, and valuable feedbacks throughout the entire duration of the thesis. I am very grateful to my friends Prakash Subedi, Dipesh Paudel, Udhyan Timilsina, Bishwa Subedi, Abhishek Bhattarai, Nirajan Pant, Jenny Maharjan, Shiva Bhattrai, Kishor Lamichhane and many more for giving me memorable time staying abroad, and guiding me and giving me suggestions throughout my studies and thesis work. Special thanks to Bikram Thapa for providing me technical guidance.

Finally, I would like to express my greatest appreciation to my parents, and of course Mitrata for love, support, inspiration and ecouragement throughout writing this thesis and my life in general.

Sandeep Kumar Shrestha
Tampere, 20th September, 2015

CONTENTS

1. INTRODUCTION 1
2. BACKGROUND 3
2.1 General agent definition 3
2.1.1 Mobile software agent 3
2.2 Why mobile agents? 5
2.3 Multi Agent System (MAS) 7
2.4 Why standardization 7
2.5 Thesis objective 8
3. HTML5 AGENT FRAMEWORK 10
3.1 Overview of HTML5 agent framework 10
3.1.1 Agent-to-agent communication in HTML5 agent framework 13
3.1.2 Agent management in HTML5 agent framework 14
3.2 Evaluation of HTML5 agent framework 16
4. INTRODUCTION TO FIPA 18
4.1 Agent communication 24
4.2 Agent management 33
5. MAPPING/COMPATIBILITY BETWEEN HTML5 AGENT FRAMEWORK AND FIPA 37
5.1 Proposed solution 38
6. PROOF OF CONCEPT/EXPERIMENTATION 42
6.1 Implementation in JADE MAS 44
6.2 Implementation in HTML5 agent framework 54
7. EVALUATION 57
8. CONCLUSIONS 58

LIST OF FIGURES

Figure 2.1 Basic mobile agent architecture [recreated from 42]. *4*
Figure 2.2 Client-server computing paradigm [recreated from 42]. *5*
Figure 2.3 Mobile agent computing paradigm [recreated from 42]. *6*
Figure 3.1 HTML5 agent life cycle [recreated from 20]..... *11*
Figure 3.2 Basic architecture of framework [recreated from 14]..... *12*
Figure 3.3 Agent-to-agent communication model [recreated from 14]. *13*
Figure 3.4 Creation of communication component [recreated from 14]..... *14*
Figure 3.5 Sending message to another agent [recreated from 14]. *14*
Figure 4.1 Agent-based system components and its interfaces [recreated from 2]..... *18*
Figure 4.2 FIPA specification categories [recreated from 3 and 5]. *19*
Figure 4.3 FIPA agent communication specification [recreated from 5]..... *21*
Figure 4.4 FIPA agent management reference model [recreated from 43]..... *22*
Figure 4.5 Syntax for FIPA-ACL message [recreated from 65]. *28*
Figure 4.6 A message [recreated from 24]. *29*
Figure 4.7 A FIPA message to transport-message [recreated from 24]. *30*
Figure 4.8 FIPA-ACL communication model [recreated from 43]. *31*
Figure 4.9 Connection between FIPA-ACL communication model and OSI reference model [recreated from 81]. *32*
Figure 4.10 FIPA specifies Agent Interaction Protocol Stack (AIPS) for Multi-agent System (MAS) interaction [recreated from 65]. *33*
Figure 4.11 Representation of agent management [recreated from 43]. *34*
Figure 4.12 Agent message transport reference model [recreated from 36]. *35*
Figure 5.1 FIPA-compliant gateway approach to make an agent platform FIPA-compliant [recreated from 3 and 75]. *40*
Figure 6.1 Outline of communication between HTML5 Agent Framework and JADE. *43*
Figure 6.2 Implementation detail for creating web application in JADE MAS [recreated from 78]. *45*
Figure 6.3 Project structure of implementation in JADE in NetBeans IDE. *46*
Figure 6.4 Steps in execution in JADE. *53*
Figure 6.5 Representation of message flow between HTML5 agent and JADE agent. *56*

LIST OF ABBREVIATIONS

AAP	April Agent Platform
ADK	Agent Development Kit
ACC	Agent Communication Channel
ACL	Agent Communication Channel
AMS	Agent Management System
API	Application Program Interface
CCL	Constraint Choice Language
CORBA	Common Object Request Broker Architecture
DF	Directory Facilitator
EPL	Eclipse Public License
FIPA	Foundation for Intelligent Physical Agents
GPL	General Public License
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JADE	JAVA Agent Development Framework
JSON	JavaScript Object Notation
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
LEAP	Lightweight Extensible Agent Platform
LGPL	Lesser General Public License
MASIF	Mobile Agent System Interoperability Facility
MAS	Multi Agent System
MTP	Message Transport Protocol
MTS	Message Transport Service
OMG	Object Management Group
ORB	Object Request Broker
RDF	Resource Description Framework
REST	Representational State Transfer
RMI	Remote Method Invocation
RPC	Remote Procedure Call
REV	Remote Evaluation
SL	Semantic Language
SOAP	Simple Object Access Protocol
TUT	Tampere University of Technology
URL	Uniform Resource Locator
XML	Extensible Markup Language

1. INTRODUCTION

The HTML5 Agent Framework [14][20][41] is developed in TUT. It is an agent-based system that demonstrates the functionality and usage of mobile agents or mobile software agents. The mobile agents in this framework are based on web technologies. The main purpose of this thesis is to "map" current implementation of HTML5 Agent Framework with agent-related standards. The term "map" here in this thesis, refers to an attempt to analyze compliance and compatibility to make HTML5 Agent Framework a FIPA-compliant system. For the standardization of mobile agents, organizations like Foundation for Intelligent Physical Agents (FIPA), Object Management Group (OMG), etc. have come forward to address standardization issues of agent-based systems. These standard bodies provide specifications and guidelines for developers of agent frameworks in creating agents for real world applications. The motivation behind FIPA is to promote agent-based technologies and the interoperability of its standards with other agent-based technologies that facilitate the end-to-end interworking of agent systems in modern commercial and industrial settings.

An agent is a computational entity capable of doing things intelligently and autonomously on behalf of its user/users to accomplish a task. Like humans, agents can co-operate and co-ordinate with each other to combine their abilities to resolve problems. Agent-based systems exhibit collaborative and co-operative problem solving behavior. But, each agent system could have been developed by different companies, at different times using different underlying agent technologies. For such scenario, standardizing agent-based technologies represents a key requirement for the commercialization of agent technology [82]. Agent systems developed by different developers or vendors differ widely in areas such as: agent management, agent capability advertisements, strategy for finding agents, agent communication language (ACL), agent dialogue mediation, message transport mechanism, and agent content language [74]. Homogeneity between agent-based systems cannot be achieved with respect to architecture specific features of agent systems. The differences among agent systems prevent interoperability and rapid growth of agent technology. Therefore, it is necessary that some aspects of agent-based systems must be standardized to achieve or promote interoperability and compatibility between heterogeneous agent systems.

The thesis is structured in the following manner: Chapter 2 contains general information about mobile agents, Multi-agent System (MAS), agent system standardization, and objective of thesis. Chapter 3 gives an overview of HTML5 Agent Framework. Chapter 4 provides introduction to FIPA. Chapter 5 includes mapping between HTML5 Agent

Framework and FIPA. Chapter 6 describes the proof of concept of this thesis. Chapter 7 draws conclusion from this thesis work.

2. BACKGROUND

2.1 General agent definition

An agent is a computational component that is capable of independent or autonomous action on behalf of its users in some environment controlling its own internal state. An agent has an ability to do certain tasks without immediate human intervention and supervision [70]. It can perform its actions with certain level of proactivity and/or reactiveness. Moreover, an agent can exhibit some level of the key attributes of learning, cooperation and mobility [42]. Some of the agent attributes discussed in [70][72] is described below:

- **Reactivity,** an ability of an agent to sense its dynamic environment and act based on it.
- **Autonomy,** an ability of an agent to make decisions over its own actions without direct intervention of humans or others and has control over its actions and internal state.
- **Proactiveness/Goal-orientedness,** an ability of an agent to take initiative and recognize opportunities in an attempt to achieve goals. It is not solely driven by events, but taking the initiative when appropriate.
- **Communication/Social-ability,** an ability of an agent to interact with other agents (and possibly humans) via some kind of Agent Communication Language (ACL) to achieve goals.
- **Cooperation,** an ability of an agent to realize that some goals can only be achieved with the cooperation with other agents.
- **Learning/Adaptivity,** an ability of an agent to learn from history and to adapt to changes with improved performance over time.
- **Continuity,** an ability of an agent to act for undefined time.
- **Mobility,** an ability of an agent to move from one machine to another in a network and continue its execution there.

2.1.1 Mobile software agent

The above mentioned attributes are some of the dimensions in which one can characterize agent. As mentioned in [72], there have been attempts to classify agent types based on above mentioned attributes. Necessary agent attributes are determined by the nature of the task that the agent should achieve. In many cases, it is not necessary to create an

agent that incorporates all the above mentioned attributes in *section 2.1* [72]. Mobile software agent or mobile agent is a self-directed application or piece of software that can migrate from one host to another in a network to accomplish the task on behalf of its user or another application [14]. Mobility is the core attribute in a mobile agent technology [72]. The mobile agent can choose when and where to migrate. Furthermore, at any arbitrary point, it can suspend its execution, transport itself to another host and resume its execution [42].

According to [42], "a mobile agent is a software entity which exists in a software environment. It inherits some of the characteristics of an agent (as discussed above in *section 2.1*). A mobile agent must contain all of the following models: an agent model, a life-cycle model, a computational model, a security model, communication model and a navigation model." Reference models for mobile agent as defined in mobile agent definition are described in [42]. The models for HTML5 mobile agent which is developed in TUT [41] are discussed in *section 3.1* of this thesis with reference to [14]. A mobile agent exists and executes in an environment. This environment can be called mobile agent environment or agent platform. Referring [42], a mobile agent environment or agent platform is a distributed software environment distributed over a network of heterogeneous computers. The mobile agent environment is built on top of a mobile agent host system.

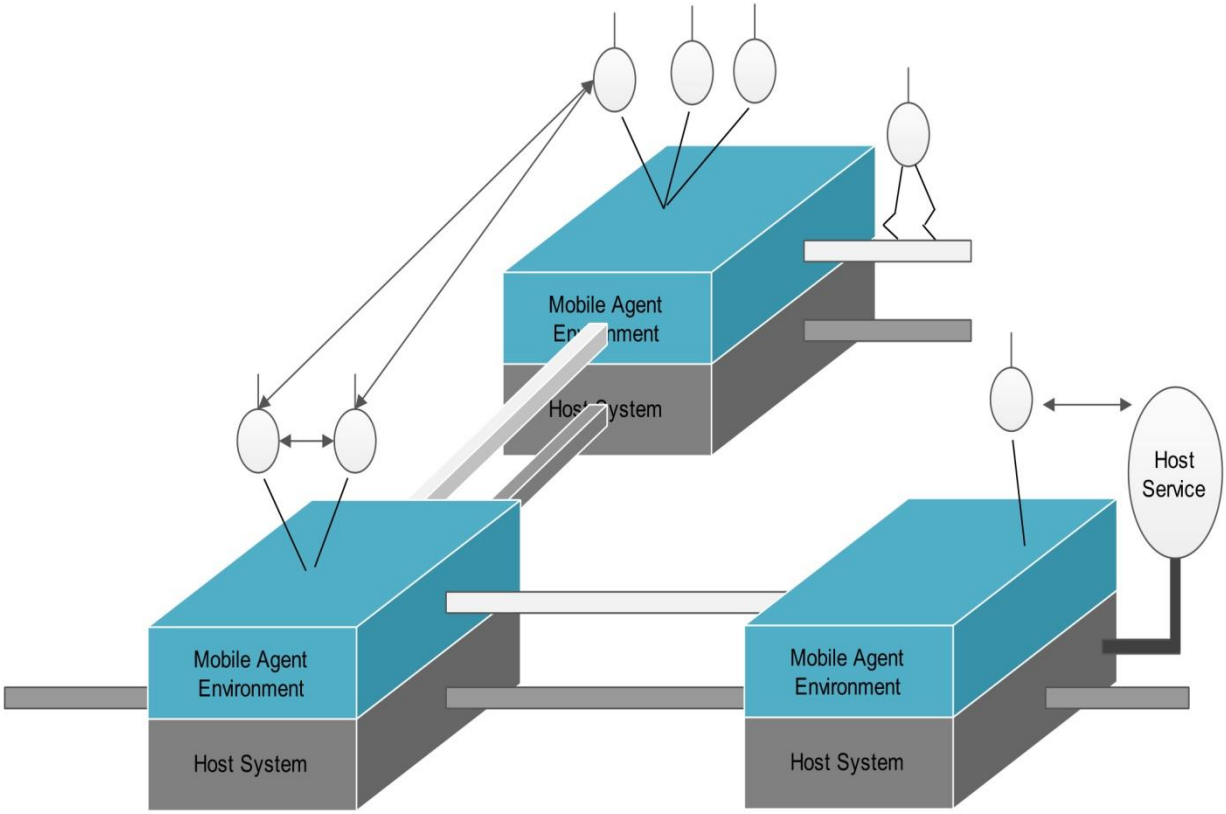


Figure 2.1 *Basic mobile agent architecture [recreated from 42].*

The above *figure 2.1* represents basic mobile agent architecture. As shown in the figure, the mobile agent environments are built on top of host systems. The small circles represent mobile agents which can travel from one mobile agent environment to another across a distributed network. Communication between mobile agents is represented by bi-directional arrows. Moreover, communication can also take place between a mobile agent and a host system service as shown in the *figure 2.1*.

2.2 Why mobile agents?

Before the emergence of mobile agents, the communication between the client and server was achieved by approaches such as message passing, Remote Procedure Call (RPC) and Remote Evaluation (REV). In client-server model, the data is queried from server over the network. The server provides a set of services and the client makes request for those services through a communication channel. When a client needs a particular service, it sends request message to the server as shown in *figure 2.2*. One of the limitations of client-server paradigm is that the client is limited to the operations or services provided at the server. If a client needs a service that a particular server does not provide, it must find out other servers that satisfy its request by sending out more messages to other servers. This behaviour may increase the inefficient use of network bandwidth. This may also increase the network traffic and may cause delays due to the involvement of more servers [72].

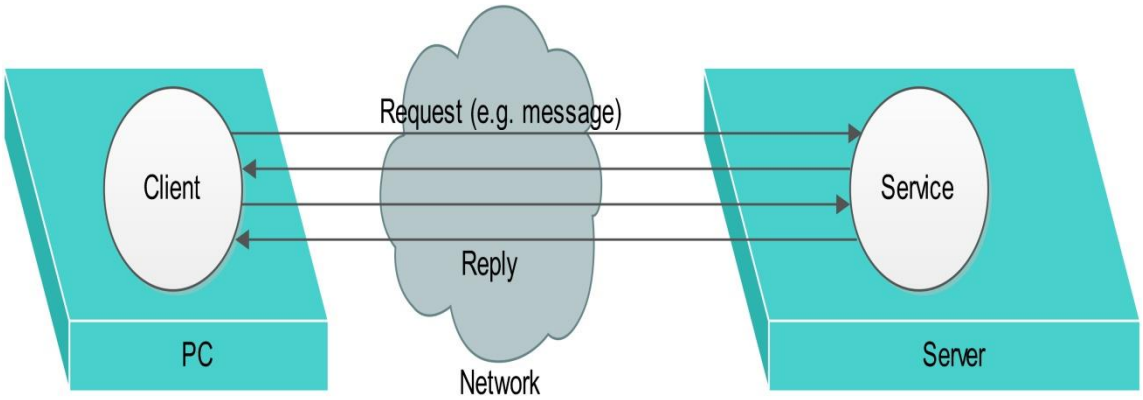


Figure 2.2 *Client-server computing paradigm [recreated from 42].*

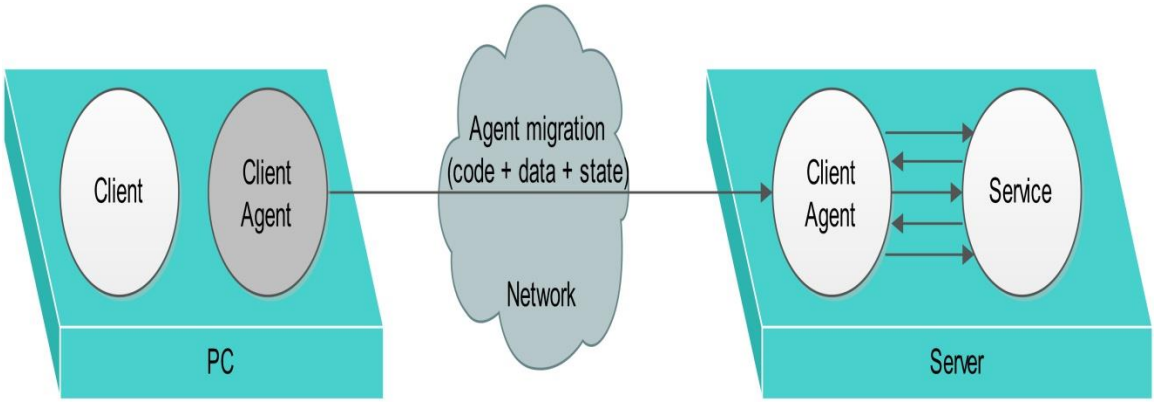


Figure 2.3 *Mobile agent computing paradigm [recreated from 42].*

In contrast, in a mobile agent computing paradigm, a mobile agent is not bound only to the system on which it begins execution. It is free to travel from one host to another host in a network to accomplish the task on behalf of its client/user. Though, it is created in one execution environment, it can carry its state, code and data with it to another execution environment in the network, where it can resume its execution [71]. So, the overall data processing mechanism is different compared to client-server model as shown in *figure 2.3*. Mobile agent is transmitted to the source of data, i.e., to the server. It processes data at the data source, rather than fetching it remotely. It performs necessary computation on behalf of its client/user in new execution environment and returns to the original location with acquired result for the client/user [14]. The main objective behind agent-based computation is to move the computation to the data source than the data to the computation allowing high performance operation [71]. Some of the benefits of using mobile agents are further discussed in [71].

Code mobility is an important aspect of mobile agent technology. Code mobility, as described in [21] is: "the capability to reconfigure dynamically, at run time, the binding between the software components of the application and their physical location within a computer network". Mobile code has the ability to transfer the state of an execution unit or mobile agent from one execution environment to another. There are two main types of migration: strong and weak. Strong migration means that the mobile agent can carry itself, its data and its execution state to a different environment. The execution of an executing unit is suspended, when transmitted to remote or destination site, and its execution is resumed there. While, weak migration means that the mobile agent can carry only itself and its data. The execution state is not transferred across the network. It only allows a mobile agent in an environment to be bound dynamically to code coming from a different environment. That means, the code can be moved and executed automatically in destination site [21][72].

2.3 Multi Agent System (MAS)

A multi agent system (MAS) is a distributed computing model. It consists of agents and their environment. A MAS is composed of a number of agents that interact with one-another to solve problems within an environment. Agents in MAS should interact for giving solutions and for solving some specific tasks. Agents act on behalf of users with different goals and motivations. So, they must have ability to cooperate, coordinate, collaborate and negotiate with each other to successfully interact in order to solve problems. Cooperation represents a general form of interaction between agents. Coordination is about organizing actions of different agents in achieving their goals. Collaboration is all about the allocation of tasks and resources between agents, and whenever conflict occurs, negotiation techniques are used to satisfy all parties. Agent communication facilitates the cooperation, coordination, collaboration and negotiation with each other, much as people do [73][76]. Agent communication standard and its objectives are further discussed in *section 4.1*. Moreover, agents in one MAS can interact or interoperate with agents in other third party MAS for cooperative problem solving. In *section 2.1*, *figure 2.1* represents a basic mobile agent architecture where a combination of host system and mobile agent environment represents a MAS. Several heterogeneous MASs can be combined together across a distributed network for distributed problem solving. Interoperability issues between these MASs are addressed by standard body like FIPA (Foundation for Intelligent Physical Agents) which is described in *section 4*.

2.4 Why standardization

MASs developed by different developers have their own architectures and technical solutions in areas like: agent management, agent capability advertisements, strategy for finding agents, agent communication language (ACL), agent dialogue mediation, and agent content language [74]. Differences in architecture and technical solution lead to heterogeneity in agent-based technology. Heterogeneity in agent-based technology isolates one agent-based system or MAS from another. This means that, there will be no interoperability between these heterogeneous systems for cooperative problem solving. There should be some mechanism to establish interoperability and compatibility between these heterogeneous systems developed by different vendors. Furthermore, without standardization process, there will be open competition between these systems and the best one always wins [4]. Standardization process allows multiple service providers to do things in the same general way and can help maximize compatibility and interoperability [83]. On the other hand, with standardization, the developers of agent system must be strongly tied up to a specific implementation of a certain specification of the agent system standards [85].

Existence of open and non-standardized MASs operating across a distributed network leads to a difficulty in locating and collaborating with agents in communities of different MASs. No agent that is designed for one of the two systems can interact with any of the agents designed for the other systems due to differences in MAS agent communication languages, architectures, and the protocols of agent communication modes [74]. Therefore, there must be some standardization mechanism to enable interoperability of MAS for cooperative problem solving. The problem before existence of standard bodies like FIPA [1] and MASIF [40] was that, agent frameworks developed by different developers of agent system were not interoperable with each other. These agent frameworks were confined to their own functionality. Agents appear in a wide range of real world applications. However, all the agent systems developed independently leads to the following problems [15]:

- **Lack of an agreed definition,** agents built by different teams has different capabilities.
- **Duplication of effort,** little or no reuse of agent architectures, designs, or components.
- **Inability to satisfy industrial strength requirements,** agents must integrate with existing legacy software and computer infrastructure.
- **Incompatibility,** agents must be able to interact and cooperate with each other.

Due to these issues, standardization is pursued. With the emergence of standardization body like FIPA, these problem statements are addressed. FIPA provides guidelines and specifications to guide developers in creating agent frameworks that are compatible and interoperable with each other. FIPA has been promoting technologies and interoperability specifications that facilitate the end-to-end interworking of agent systems in modern commercial and industrial settings. Further description about FIPA is presented in section 4 of this thesis.

2.5 Thesis objective

The main objective of this thesis is to map HTML5 Agent Framework developed in TUT [41] with FIPA standards. So, the major challenge of this thesis will be to analyse compliance and compatibility between HTML5 Agent Framework and FIPA compliant systems. FIPA specific standards and specification will be analysed and compared with HTML5 Agent Framework in later part of this thesis. Recently, many researchers have contributed their effort in transforming agent thechnology into practice to promote agent technology [16]. Evolution in concept of agent technology and interoperability has introduced the concept of Agentcities. As described in [17][18], Agentcities is a worldwide initiative to create a global, open, dynamic, intelligent, heterogeneous network of

agent platforms and services to achieve user and business goals. The ultimate goal of Agentcities is open deployment environment for advanced agent based services such that agents running on different platforms that are owned by different organisations can interact. Agentcities is based on the principles of [17][18]:

- **Consensual standards,** communication and interaction in the network will be based on standards available, such as, FIPA and W3C.
- **Open source,** commercial technologies are not discouraged, but Agentcities promotes open source implementation to ensure free and open access to the network.
- **Open access,** any organization or individual can set up their own Agentcities in the network to host their own agent services, provide access to them and access those deployed by others.
- **Shared resources,** any organization or individual are encouraged to add their own services to extend the utility and diversity of the services available to the agent community.

3. HTML5 AGENT FRAMEWORK

Research paper [20] and thesis work [14] are used as main references in this section. [20] describes the latest version of HTML5 Agent Framework. In HTML5 Agent Framework, an agent is implemented as an HTML5 application that can run in two modes: with a user interface inside a browser and in a headless mode, that is, without a user interface, in an environment called Agent Server. Agent Server represents agent environment in server-side and browser represents agent environment in client-side. The state of an agent is saved and transferred during the migration between browser and server. The agent can continue its execution even if it is in server and the running agent can be retrieved back later to the server [14][20].

3.1 Overview of HTML5 agent framework

The implementation of the HTLM5 Agent Framework consists of two parts: HTML5 Agent Framework and Agent Server. Agent works with user interface inside browser or in headless mode in an Agent Server. The HTML5 Agent Framework uses mobile agent paradigm for transferring agent and its state from browser to server, server to browser and server to another. Moreover, code-on-demand paradigm [21] is used for getting the static files of the mobile agent when agent travels in network [14]. HTML5 agent in [14][20] consists of two parts [19]:

1. User interface is defined in HTML, CSS and image files.
2. JavaScript files describing the executable content.

HTML5 agents and core functionalities of the system are implemented in JavaScript and executed inside the client's web browser. In HTML5 agent framework, the core components of Agent Server are: HTTP server and a virtual machine executing JavaScript. These components are implemented using node.js [46] technology. The Agent Server has two main functions [20]:

1. Implement execution environment that is compatible enough with the browser.
2. Simple management functionality for agents.

Referring [42], a mobile agent must have agent model, life cycle model, computational model, navigation model, communication model, and security model. Current implementation of HTML5 agent framework [14] has complete implementation of agent model, life cycle model and computational model. Navigation model, communication model, and security model are only on their initial stage.

1. Agent model
   Agent model in [14] includes the management of the inner state of the agent.

2. Life cycle model
   During a life cycle, the agent may visit several browsers and Agent Servers. *Figure 3.1* describes the life cycle of an HTML5 agent.

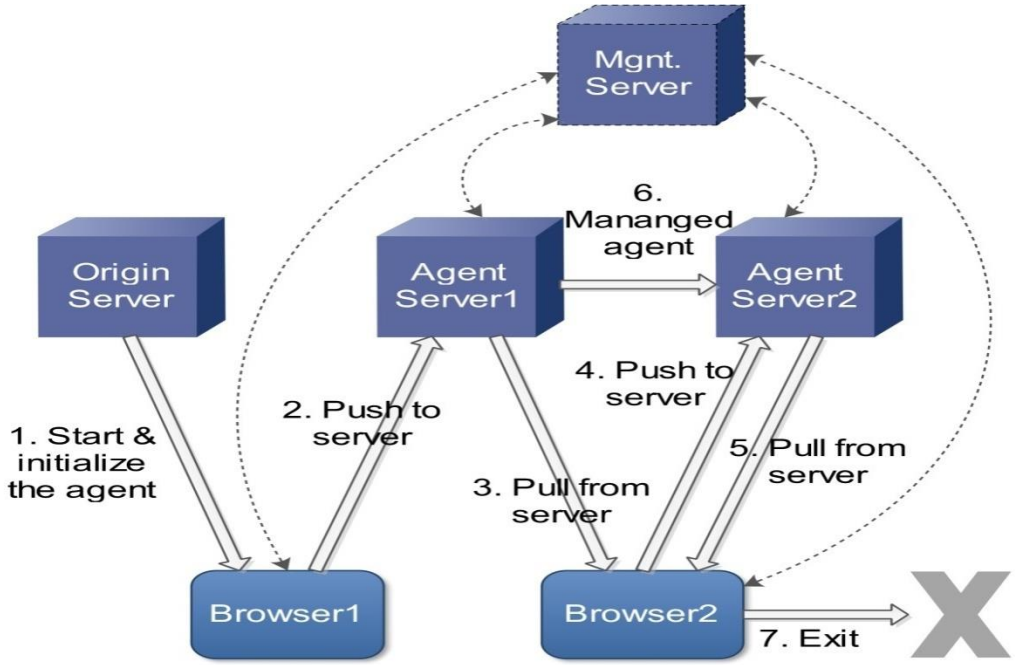


Figure 3.1 *HTML5 agent life cycle [recreated from 20].*

The instance of an agent is created when it is downloaded from Origin Server. The task of Origin Server is to host applications and it is similar to an ordinary web server. After the download and initialization, the executing agent can move to an Agent Server in order to continue its execution and back to a browser again. Both Origin Server and Agent Server are HTTP servers that can be accessed with HTTP requests. The dashed box "Mgnt. Server" and the dashed arrows represent optional management functionality [20].

3. Computational model
   Computational model [14] represents how mobile agent runs. HTML5 mobile agent runs in both browser and server. The agent framework is based on event handlers and executable contents are implemented in JavaScript.

4. Navigation model
   Navigation model [14] of the HTML5 Agent Framework consists of the configuration file that is downloaded with the agent. It includes only connections to the

one Agent Server and the Origin Server of the agent. It also includes the serialization and transfer management of the agent.

5. Communication model
   Communication model [14] includes communication with the user, Agent Server, and another agent application. Communication with user is done through user interface in web browser. HTML5 agent creates separate communication component which is used as a message passing pipe through Agent Server to another agent for agent-to-agent communication. Three cases for agent-to-agent communication are mentioned in [14]:
   - Both agent applications are in separate browsers.
   - Both agent applications are in agent server.
   - One agent is in browser and another is in server.

6. Security model
   Security model in [14] rely on standard security mechanisms of HTML5 applications in browser.

Figure 3.2 shows basic architecture of the HTML5 agent framework and its relationship to application specific implementations. Agent part of the architecture consists of a generic agent and an application agent. In server, agent can be accessed through generic interface provided by generic agent. In browser, agent can be accessed through application specific user interface [14].

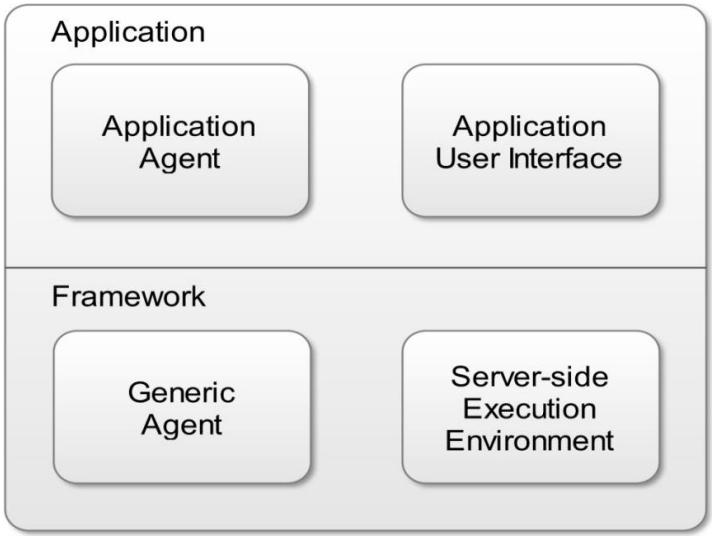


Figure 3.2 *Basic architecture of framework [recreated from 14].*

Generic agent is the base class for each agent application. It provides the generic parts o

of the agent to all agent applications. Generic agent is never instantiated but used only for providing the generic parts of the agent to all agent applications. Generic agent is also responsible for preserving the agent state. Application agent is the concrete implementation of the application that can travel between browser and server [14].

3.1.1 Agent-to-agent communication in HTML5 agent framework

The current implementation of agent-to-agent communication is generic. The implementation of communication component in agent framework is not connected to the application. It can be used with any application and the application does not need to know how the information is passed through the network. However, application has to define application specific namespace to enable application specific communication [14]. Any application that knows application specific namespace can join to namespace and capture all messages. For agent-to-agent communication, node.js [46] module socket.io [47] is used to enable real time communication between agents. Agent server is needed between two agents for agent-to-agent communication. It is not possible to send direct messages from one browser to another browser in the current Web without a server. Information between agents is sent in JSON (JavaScript Object Notation) strings over the network. Agent-to-agent communication in HTML5 agent framework is shown in *figure 3.3*.

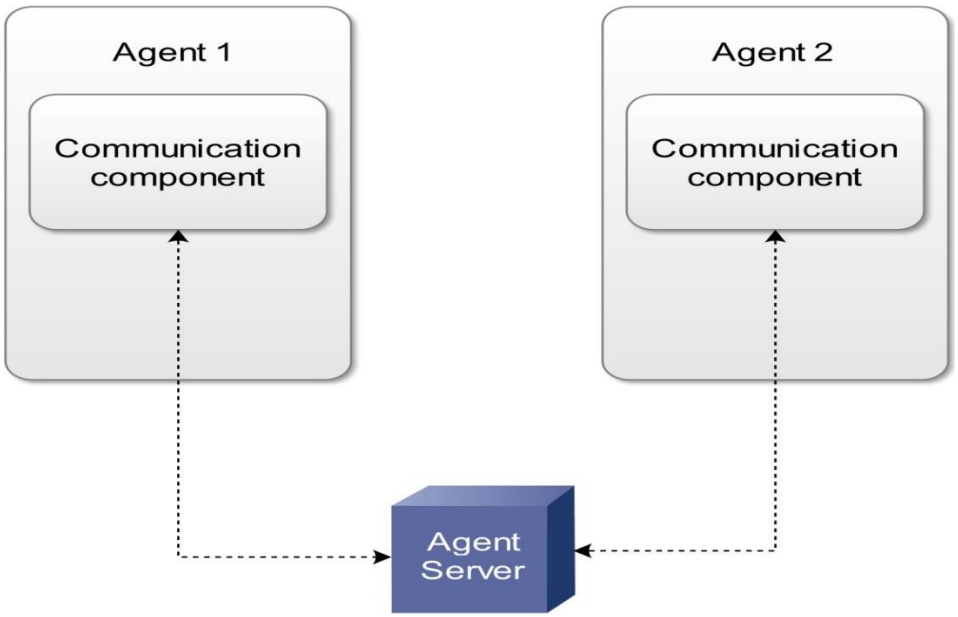


Figure 3.3 *Agent-to-agent communication model [recreated from 14].*

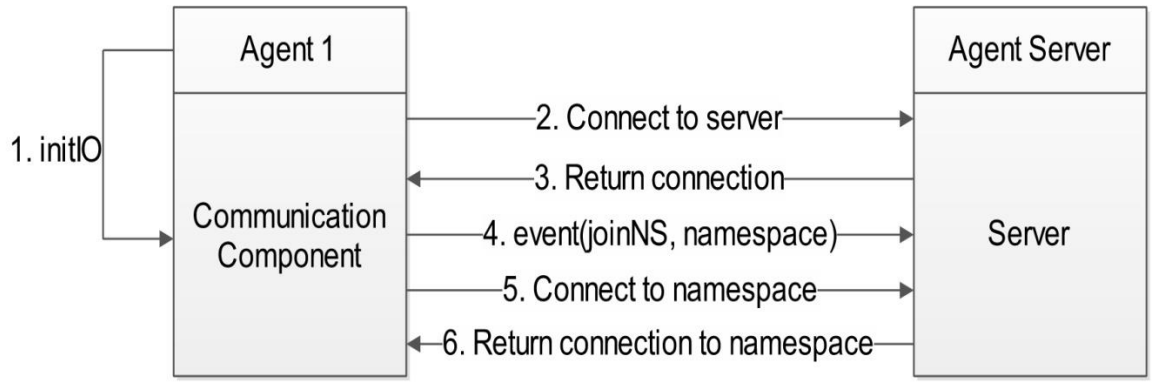


Figure 3.4 *Creation of communication component [recreated from 14].*

Figure 3.4 represents initialization of connection to specific namespace in Agent Server. Whereas, *figure 3.5* shows representation of how message is sent from one agent to another agent connected to same namespace.

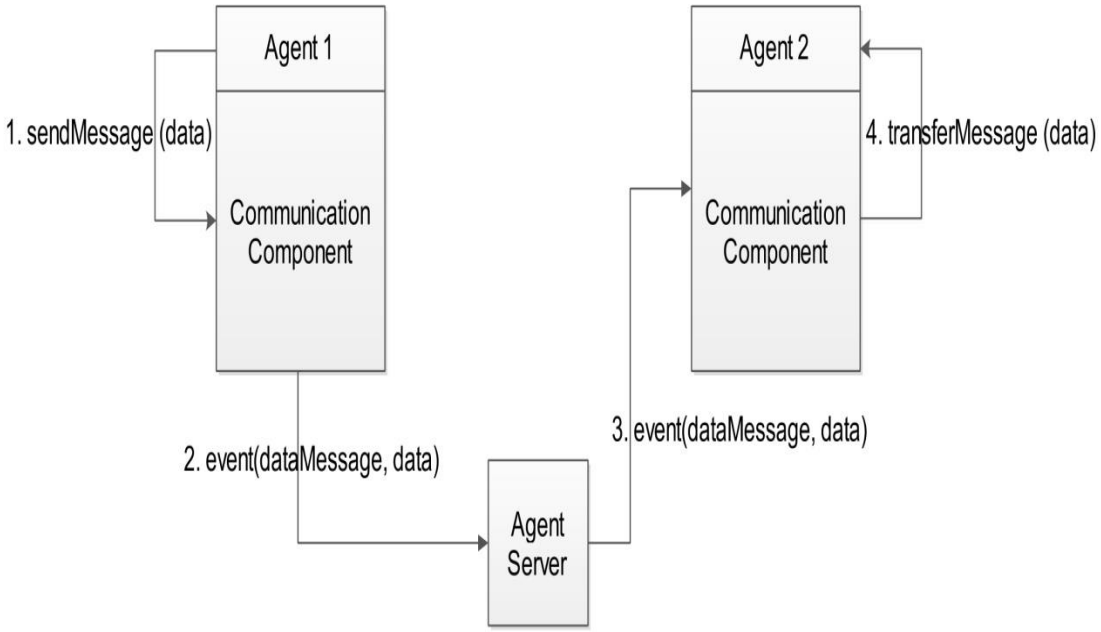


Figure 3.5 *Sending message to another agent [recreated from 14].*

3.1.2 Agent management in HTML5 agent framework

Referring [14], agent management in HTML5 Agent Framework is implemented in Agent Server. It keeps information about agents that are currently running on that Agent Server. Moreover, agent specific information is shown to the user in list view which contains URL(s) of JavaScript file and agent ID. The implementation of agent management in [14] needs to be re-factored to make it more scalable and to add new features such that management of agents in server is easier. In future, more information about

agents are needed, such as, description of the agent, configuration of agent, creation, registration, communication, migration, and retirement of agents.

Implementation of Management Server for agent management is discussed in [20]. This server represents an optional management functionality in the agent framework that allows external entities to control agents. The Management Server in [20] implements a REST (Representational State Transfer) interface for both the mobile agents and a control application. The control application can be an application run by a human or an autonomously running application. The management API for agents in Management Server consists two kinds of REST calls: "ImHere" message is sent when agent arrives in a new location, and "Status" message is sent to the Management Server on regular basis after each work [20].

As described in [20], an agent after starting in a new location sends **"ImHere"** message to the Management Server. For example:

```
Management/ImHere(PUT)
(Payload example)
From browser: {"id":"392041","client":"xhost"}
From server:
{"id":"392041","server":"http://ubuntu:8891"}
```

After each execution, the agent sends **"Status"** message to the Management Server on regular interval.

```
Management/Status(PUT)
Payload example:{"id":"392041","status":"Clock is 63"}
```

The list of agents can be queried when control application makes a GET request to **/Agents** it gets a list of agent IDs as a response.

```
GET http://host/Agents
```

For example, the response can be:

```
[846820, 920231, 934582]
```

Detailed information about a specific agent can be retrieved with:

```
GET http://host/Agents/846820
```

The response includes information about the location and status of the agent.

To support additional features, functionalities like suspension of agent, termination of agent, creation of agent, resource management, etc., can be added to current agent management implementation. The current implementation of agent management is confined to status and location of agents. Future agent management implementation in agent based system requires more dynamic functionalities in addition to status and location specific functionality.

3.2 Evaluation of HTML5 agent framework

The HTML5 Agent Framework [14][20] is based on the mobile agent and code on demand paradigms. However, the implementation in agent framework is in initial stage. Simple communication and navigation model exist in current implementation [20] of HTML5 agent framework. In the current implementation, it is assumed that agent will be moved from server to browser and it will be uploaded from browser to server. In future, it may require that information from different agent servers may be needed, such that, it is possible to communicate with agents in other agent servers. So, there should be a mechanism of sending messages to other agent servers which could then pass it to relevant agents. In the current implementation, any agent that knows application specific namespace can initialize connection to specific namespace in agent server. Message can be sent to another agent connected to the same namespace. Moreover, the current implementation does not fully support flexible code mobility. The code of the agent is transferred as URLs instead of binding the code to the agent state transferred [14]. In future, agent technology requires flexible code mobility [21] where mobile agents are capable of reconfiguring dynamically at run time. Mobile agents must be capable to bind software components of the application and their physical location within a computer network dynamically. So, there are places for improvement in current implementation of agent framework to make it interoperable in heterogeneous agent environment. HTML5 agent framework has its own benefits and weaknesses. Some of the benefits of HTML5 agent framework can be listed as below [14]:

1. Framework does not need installation of separate software or environment beside the web browser.
2. HTML5 agents run natively in web browser hence plug-ins are not needed.
3. Used Web technologies are well known and standardized.
4. HTML5 applications already have strong application ecosystem.

And the weaknesses are [14]:

1. Lack of proper security model.
   The current implementation is lacking proper security model. Whoever can send a HTML5 agent to the Agent Server and similarly retrieve HTML5 agent from Agent Server. So, the future implementation needs to define security issues to

protect the framework from intruders. On this, further research is ongoing in TUT [84].

2. Lack of standardization in agent-to-agent communication.
   The current implementation of agent-to-agent communication is very generic and does not follow any available standards. However, it is near to Message-Oriented Middleware (MOM) approach [14]. Use of existing standard like FIPA specifications for agent-to-agent communication can be benefit, because it is easier to rationalize the use of system based on existing well known standards than on self-made standards [14].

3. Inflexible navigation model.
   In HTML5 agent framework [14], agent migration is user defined. Agent does not make decision to move from server to browser or browser to server autonomously. Mobile agents as autonomous computational entity must be able to define when and where they will migrate. But, the configuration of mobile agent in [14] demands all the hosts that agents need to know in configuration file to be defined explicitly. This prevents flexible mobility of agents in network to search necessary information in order to accomplish the task on behalf of user. However, the management functionality described in [20] fixes part of this limitation reported in [14].

4. INTRODUCTION TO FIPA

The Foundation for Intelligent Physical Agents (FIPA) is an IEEE Computer Society standards organization for agents and multi-agent systems [1]. It was officially accepted by the IEEE as its eleventh standards committee on 8th June, 2005. The motivation behind FIPA is to promote agent-based technologies and the interoperability of its standards with other technologies that facilitate the end-to-end interworking of agent systems in modern commercial and industrial settings.

Originally, FIPA was a Swiss based non-profit association registered in Geneva, Switzerland in 1996 [1][2]. It facilitates international collaboration of the member organizations. The members are companies and universities actively participating in the field of agent technology. The main purpose of FIPA was to produce specifications for heterogeneous agent based systems [1]. Developers of agent systems can use the basic agent technologies specifications produced by FIPA to build complex systems with a high degree of interoperability between heterogeneous agent systems in modern commercial and industrial settings. FIPA specifies the interfaces of the different components in agent based systems environment with which an agent can interact. These interfaces can be humans, other agents, non-agent software, and the physical world [2] as shown in the *figure 4.1*.

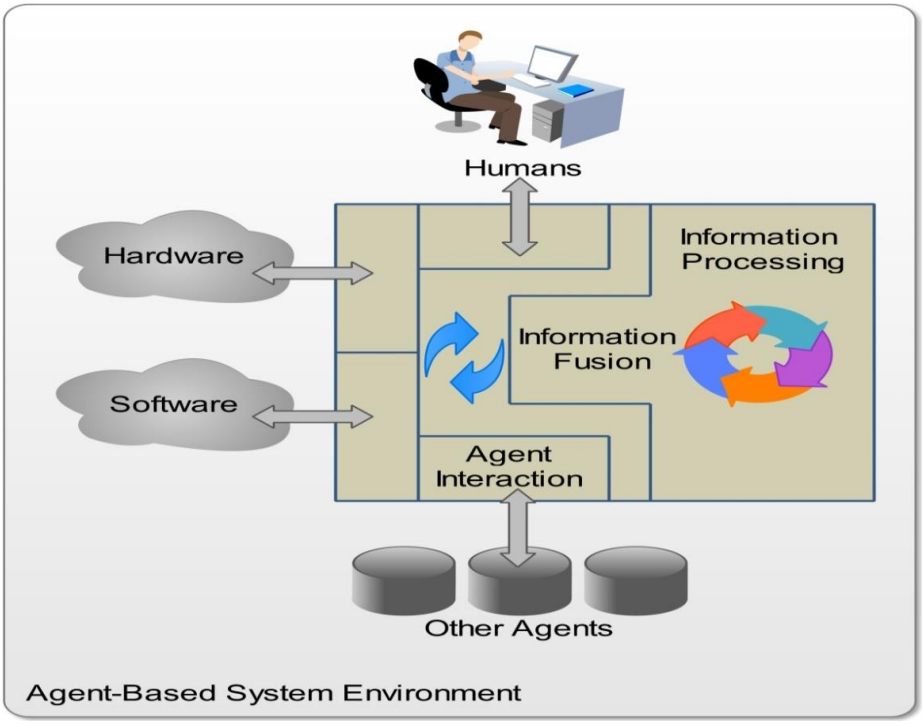


Figure 4.1 *Agent-based system components and its interfaces [recreated from 2].*

FIPA specifications do not specify what should be the internal architecture of agents, nor do they attempt to describe how agent system developers should implement agent-based systems. They just provide interfaces through which heterogeneous agents can communicate with each other in agent-based systems [3]. FIPA's principle objective is to define standards for agent-based system environment composed of heterogeneous agents built by different developers. It focuses on interoperability and compatibility between different agent systems built by different developers. According to [2], FIPA produces two kinds of specifications:

- **Normative specifications** talk about FIPA reference model for agents, agent communication language and agent/software integration (it specifies how agents may interact with non-agent based software). They define an agent's external behavior and ensure interoperability with other FIPA specified agent systems.
- **Informative specifications** comprise of guidelines on how FIPA technology can be applied in different application domains.

FIPA specifications can be divided into five categories [3]: Applications, Abstract Architecture, Agent Communication, Agent Management, and Agent Message Transport as shown in *figure 4.2*. Abstract architecture, agent communication, agent management, and agent message transport specifications are defined as normative specifications. Whereas, application specification is defined as informative specification in FIPA [2].

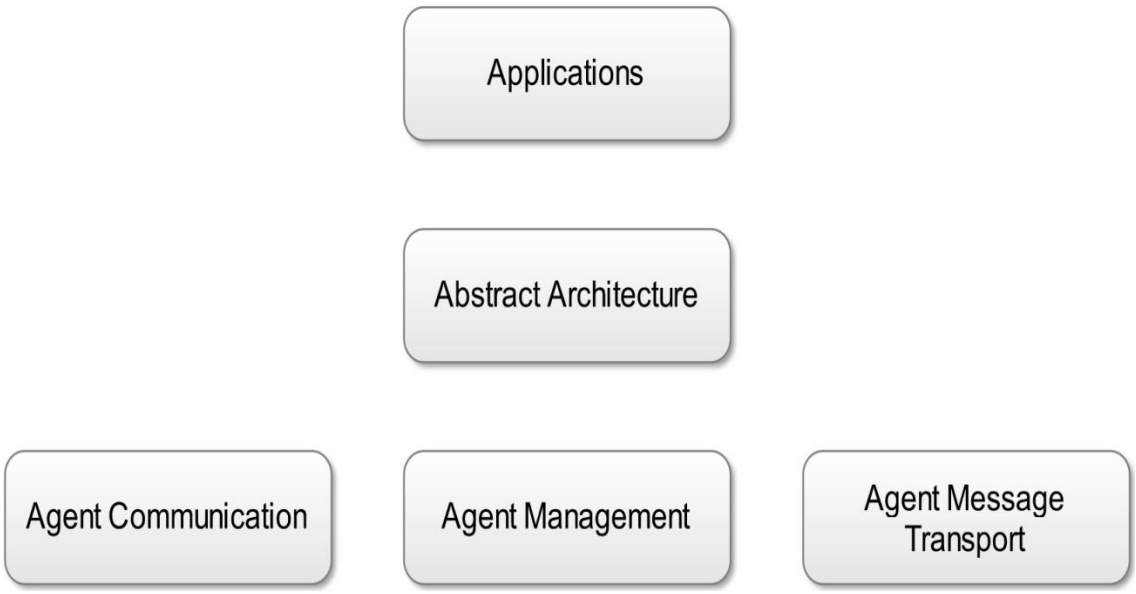


Figure 4.2 *FIPA specification categories [recreated from 3 and 5].*

1. Applications Specification
   FIPA application specification specifies how normative specifications should be applied in different application domains. [2] and [3] have given examples of four agent-based applications that contain case scenarios of agent-based system:
   - Personal Travel Assistance
   - Personal Assistant
   - Audio/Video Entertainment & Broadcasting
   - Network Management & Provisioning

2. Abstract Architecture Specification
   The FIPA abstract architecture specification [24] provides a framework which defines services necessary to enable interoperability between different agents or agent systems. If two or more agent systems use different technologies to achieve some functionality, it is necessary to identify common characteristics between these various approaches. By identifying relationships or common characteristics of the fundamental elements of the architecture, it is easier to build interoperable agent system. FIPA abstract architecture specification identifies architectural abstractions that can be formally related to any valid implementations [4].

3. Agent Management Specification
   The FIPA agent management specification [30] provides the framework, within which FIPA agents can exist, operate and be managed [2][3][4]. It defines functionality for the creation, registration, location, communication, and retirement of agents. It defines agent platform reference model containing such capabilities as white and yellow pages, message routing, and life cycle management [2]. The FIPA agent management consists of following components which will be described further later in this chapter:
   - Agent Management System (AMS)
   - Message Transport Service (MTS)
   - Directory Facilitator (DF)

4. Agent Message Transport Specification
   The FIPA agent message transport specification deals with the delivery and representation of messages on top of different network transport protocols, including wire line and wireless environments [3]. It contains specification for transport of message between agents. A message at the message transport level consists of a message envelope and a message body. The message envelope consists of specific transport requirements and information that will be used by the Message Transport Service (MTS) to handle and route messages on each Agent Platform (AP). Whereas, the message body contains actual information or mes-

sage payload that is expressed in FIPA ACL (Agent Communication Language) [3].

5. Agent Communication Specification
   The FIPA agent communication specification deals with Agent Communication Language (ACL) messages, message exchange interaction protocols, speech act [49] theory-based communication acts, and content language representations [5].

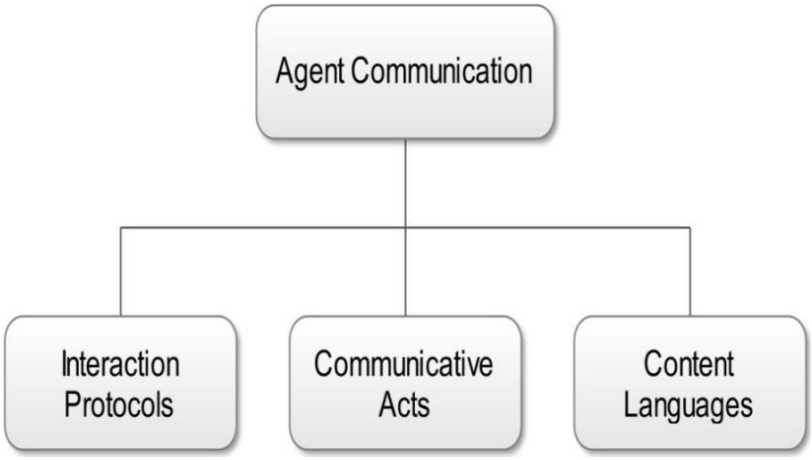


Figure 4.3 *FIPA agent communication specification [recreated from 5].*

The agents communicate with each other by means of well-defined communication language called FIPA-ACL. The FIPA-ACL is based on speech act theory; messages are actions or communicative acts (also called the "*performative"*) indicating what the sender intends to achieve by sending the message. For instance, if the performative is REQUEST, the sender wants the receiver to perform an action, if it is INFORM the sender wants the receiver to be aware of a fact. The objectives behind standardizing the FIPA-compliant ACL messages are [6]:

- To ensure interoperability between different agents existing in an agent platform or multiple agent platforms by providing standard set of ACL message structure.
- To provide a well-defined process for maintaining this set.

Referring [2], the specification also provides the normative description of a set of high level interaction protocols, including requesting an action, query, contract-net and several kinds of auction specifications. Agent communication specifications and its standards are further discussed in *section 4.1*.

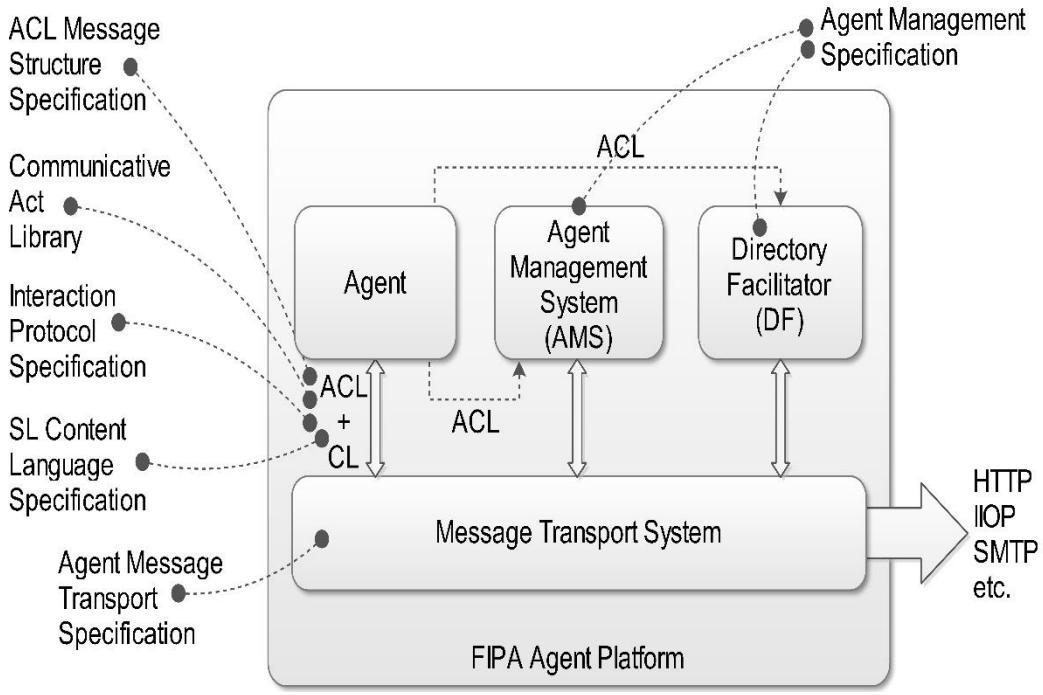


Figure 4.4 *FIPA agent management reference model [recreated from 43].*

Agent life cycle management, message transport, message structure, inter-agent interaction protocols, ontologies, and security are defined within the scope of FIPA [43]. The *figure 4.4* represents the basic structure of agent system compliant to FIPA. All the logical components of FIPA agent management reference model are described in *section 4.2*. Here is a list of major publicly available implementations of agent platforms that comply with FIPA Specifications [7]:

1. Agent Development Kit (ADK)
   The Agent Development Kit [7][37] is a mobile agent-based development platform that allows developers of agent system to build reliable and scalable industrial strength applications. The ADK uses a reliable, lightweight runtime environment based on Java that features dynamic tasking, JXTA (Juxtapose) based P2P architecture with XML (Extensible Markup Language) message-based communication that supports FIPA and SOAP (Simple Object Access Protocol), JNDI (Java Naming and Directory Interface) directory services. These ADK features allow Java system developers to easily build, deploy and manage secure, large-scale distributed solutions that support interoperability regardless of location, agent platform environment or protocol, allowing an adaptive, dynamic response to changes. The ADK runs on any environment supporting Java 2 Standard Edition version 1.3.1. It requires commercial license to access ADK. However, free research license is available for selected projects.

2. April Agent Platform (AAP)
   The April Agent Platform [7][38] is a lightweight and powerful solution for developing agent-based systems that comply with FIPA agent platform specification. The AAP requires the April programming language and the InterAgent Communication System (IMC) to be installed, and runs either on Linux, UNIX or Windows. It provides many features to develop and deploy agents and agent platforms. The AAP can be accessed using GPL (General Public License). The GPL [8] is free software license which allows end users the freedoms to use, study, share, and modify the software.

3. Comtec Agent Platform
   The Comtec Agent Platform [7] is an open source, free implementation of FIPA agent communication, agent management, agent message transport and some of the applications. It runs on JDK 1.2 or higher, and can be accessed using GPL.

4. FIPA-OS
   The FIPA-OS [7][39] is the first Open Source implementation of the FIPA standard. Developers around the world have contributed their part to numerous bug fixes and upgrades. The FIPA-OS is one of the most popular agent systems implementation that complies with FIPA specifications. It is implemented using Java, and requires Java virtual machine to implement FIPA-OS. It requires Eclipse Public License (EPL) to access FIPA-OS. The EPL [9] is Open Source software license used by Eclipse Foundation for its software.

5. Grasshopper
   Grasshopper [7] is a Java-based mobile intelligent agent platform. It is a universal agent platform based on OMG-MASIF (Mobile Agent System Interoperability Facility) [40] and FIPA specifications. MASIF is also a standard for mobile agent systems which has been adopted as an OMG (Object Management Group) technology [40].

6. JACK Intelligent Agents
   JACK Intelligent Agents [10] is a framework in Java for multi-agent system development. It was built by Agent Oriented Pty. Ltd. (AOS), based in Melbourne, Australia. The AOS's aim is to provide a platform for commercial, industrial and research applications.

7. JADE (JAVA Agent Development Framework)
   JADE [7][12] is a FIPA compliant software framework to develop interoperable intelligent multi-agent systems. It is an Open Source platform for P2P agent based applications. It is implemented in version 1.2 of Java. It can be accessed

using Open Source license Lesser General Public License (LGPL). The LGPL [11] is a free software license published by the Free Software Foundation (FSF).

8. JAS API (Java Agent Services)
   The Java Agent Services [7] is an implementation of the FIPA Abstract Architecture within the Java Community Process (JCP) [13] initiative and is intended to form the basis for creating commercial grade applications based on FIPA specifications.

9. LEAP (Lightweight Extensible Agent Platform)
   The LEAP [7] is a development and run-time environment for intelligent agents. It aims to become the first integrated agent development environment capable of generating agent applications in the ZEUS environment and executing them on run-time environments derived from JADE. The advanced features of ZEUS and the lightweight and extensible properties of JADE add benefits to LEAP agent platform.

10. ZEUS
    ZEUS [7] is an Open Source agent system implemented in Java. It is developed by BT Labs and can be considered a toolkit for developing interoperable multi-agent applications. ZEUS uses the latest Swing GUI components, and runs on any platform that has a JDK2 virtual machine installed. It has been successfully tested on Windows 95/98/NT4 and Solaris platforms.

Referring [22], the following FIPA-compliant networks are publically available:

1. **Agentcities** [17][18]**,** an initiative to create a next generation Internet based upon a worldwide global network of services that use the metaphor of a real or a virtual city to cluster services [22].

2. **FIPA-NET,** an early attempt to create multiple inter-linked FIPA agent platforms, whose activities are now continued in Agentcities [22].

4.1 Agent communication

Agents communicate in order to achieve their goals or goals of an agent platform in which they reside. Agents communicate to coordinate their behaviour and actions. It helps in creating systems that are more coherent. Agent platform provides necessary computational infrastructure for interactions between agents to take place. The computational infrastructure will include protocols for agents to communicate and interact [25]. An agent based system is composed of multiple autonomous agents. Agents must have some communicative abilities to cooperate and coordinate with other agents. The

aim of an agent system is to achieve goals that are difficult to achieve by the functionality of an individual agent. So, in an agent system, knowledge sharing and exchange of information between different agents are important [26]. Referring [26], for agents to communicate with other agents, they must be able to:

- **Deliver and receive messages,** at the physical level, communication between agents must take place over agreed physical and network layers to be able to deliver and receive strings or object that represent messages.
- **Parse the messages,** at the syntactic level, agents must be able to resolve messages to correctly decode to its parts, such as: content of the message, language, and sender.
- **Understand the messages,** at the semantic level, the parsed symbols must be understood in the same way, that is, the ontology describing the symbols must be explicitly expressed or shared and must be accessible to be able to decode the information contained in the message. According to Thomas Gruber, the term ontology refers to "explicit specification of conceptualization", which means that, an ontology is a description of the concepts and the relationships that can exist for an agent or a society of agents [27]. An ontology must be agreed and understood among the agent community (or at least among its part) in order to enable each agent to understand content of messages from other agents [28]. An agent is able to communicate only about facts that can be expressed in some ontology [28]. The ontology must be expressed explicitly in open multi-agent systems, where agents developed by different agent system developer may need to enter into communication to enable integration. So, for such environment, it is necessary to have standard mechanism to access and refer to explicitly defined ontologies. Translation between ontologies is needed when multiple ontologies are used in a system for agents to be able to communicate [28].

The physical level and the syntactic level mentioned above are well standardized in the FIPA specifications, like "Agent Management Specification" [30] and "Agent Communication Language Specification" [29]. For the semantic level, other FIPA standards exists named FIPA Semantic Language Content Language Specification [31] that describes content language and FIPA Ontology Service Specification [32] that describes usage of ontologies [26].

For communication between agents FIPA has specified Agent Communication Language [29] specification. According to [29], a FIPA ACL message contains several message parameters. Depending on situation, required parameters may vary for effective agent communication. The only mandatory parameter in all ACL messages is the performative parameter (communicative act). However, most ACL messages will also contain sender, receiver and content parameters. The full set of FIPA ACL message

parameters is shown in *Table 4.1*. A number of communicative acts (performatives) [33] are also specified by FIPA, which is shown in *Table 4.2*.

Table 4.1. FIPA ACL message parameters [adopted from 29 and 43].

Parameter	Description
performative	Defines what action the message performs.
sender	Defines the identity of the initiator of the message.
receiver	Defines the identity of the recipient of the message.
reply-to	Defines the recipient of message reply.
content	Defines the content of the message.
language	Defines the language in which the content parameter is expressed.
encoding	Defines the specific encoding of content language expression.
ontology	Defines the context of content.
protocol	Defines the interaction protocol that the sending agent employs
conversation-id	Defines the conversation identifier to identify the ongoing sequence of communicative acts that together form a conversation.
reply-with	Defines an expression that will be used by the responding agent.
in-reply-to	Defines an expression that references an action to which this is a reply.
reply-by	Defines a time and/or date by which the sending agent would like to receive a reply.

Table 4.2. FIPA communicative act [adopted from 33 and 43].

Accept-proposal the action of accepting a previously submitted proposal.	*Propagate* the receiver treats the embedded message as sent directly to it by the sender, and must identify the agents denoted by the given descriptor and send the received *propagate* message to them.
Agree agree to perform some action, possibly in future.	*Propose* submit a proposal to perform a certain action, given certain preconditions.
Cancel cancel some previouslsy requested action.	*Proxy* the receiver must select target agents denoted by a given description and to send an embedded message to them.
Call for Proposal make a call for proposal to perform a given action.	*Query-if* ask another agent whether a given proposition is true.
Confirm inform a receiver that a given proposition is true, where the receiver is known to be uncertain about the proposition.	*Query-ref* ask another agent for the object referred to by a referential expression.
Disconfirm inform a receiver that a given proposition is false.	*Refuse* refuse to perform a given action.
Failure inform another agent that an action was attempted but failed.	*Reject-proposal* reject a proposal to perform some action during a negotiation.
Inform inform a receiver that a given proposition is true.	*Request* a sender requests a receiver to perform some action.
Inform-if a macro action for the agent of the action to inform the recipient whether or not a proposition is true.	*Request-when* a sender requests a receiver to perform some action when some given proposition becomes true.
Inform-ref a macro action for sender to inform the receiver the object which corresponds to a descriptor, for example, a name.	*Request-whenever* a sender requests a receiver to perform some action as soon as some proposition is true and thereafter each time the proposition becomes true again.
Not-understood informs a receiver that sender did not understand the message that the receiver sent to it.	*Subscribe* a persistent intention to notify the sender of a value of a reference, and to notify again whenever the object identified by the reference changes.

Communicative act (also called performative) is based on speech act [49] theory. It represents the intention of the sender of the message. For example, when an agent sends an INFORM message it wishes the receiver(s) to become aware about a fact (e.g. (INFORM "today it's snowing")). Similarly, when an agent sends a REQUEST message it wishes the receiver(s) to perform an action. FIPA has defined 22 communicative acts and each one has a well-defined semantics [12]. A FIPA ACL message example recreated from [65] is shown below:

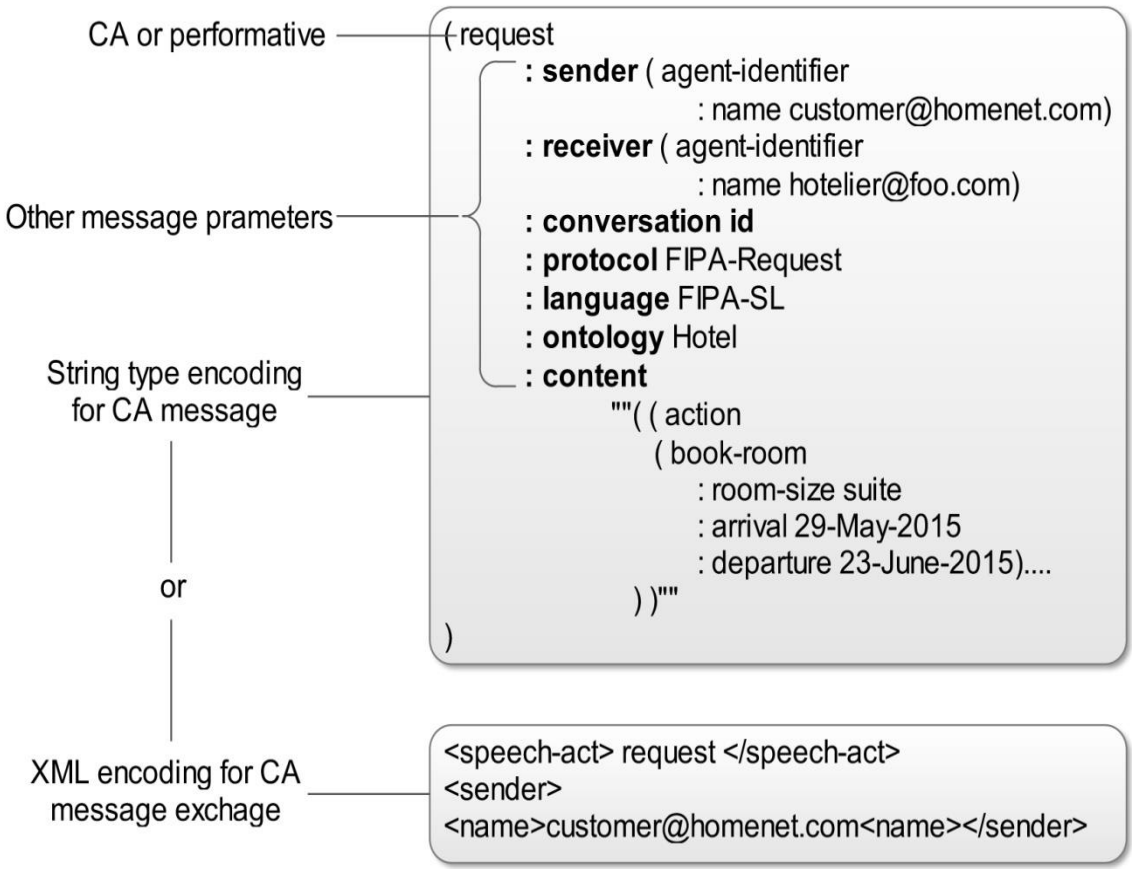


Figure 4.5 *Syntax for FIPA-ACL message [recreated from 65].*

There are two fundamental aspects of message communication between agents [24]:

1. Message Structure
   The FIPA ACL message structure consists of type of communicative act, identity of sender and receiver as well as the ontology and interaction protocols of the message. The structure of a message is a key-value-tuple (KVTs) which is written in an agent communication language. The content of the message is expressed in a content-language, such as KIF (Knowledge Interchange Format) [34], SL (Semantic Language) [31], CCL (Constraint Choice Language) [50], or RDF (Resource Description Framework) [51]. The content-language defines the

grammar and associated semantics for expressing the content of a message. Ontology defines the vocabulary and meaning of the terms and concepts used in content expression. The sender and the receiver agents are identified by agent-names. Interaction protocols [35] specify communication patterns of agents. Some of the FIPA defined interaction protocols are as follow: FIPA-Request [52], FIPA-Query [53], FIPA-Request-When [54], FIPA-Contract-Net [55], FIPA-Iterated-Contract-Net [56], FIPA-Auction-English [57], FIPA-Auction-Dutch [58], FIPA-Brokering [59], FIPA-Recruiting [60], FIPA-Subscribe [61], and FIPA-Propose [62].

Figure 4.6 *A message [recreated from 24].*

2. Message Transport
   When a message is sent it is encoded into a payload. Payload encodes a message into another representation making it suitable for transport over the message transport. Moreover, an appropriate envelope is created which includes the sender and receiver transport-descriptions. The envelope may also contain additional attributes such as the encoding representation and data related security. The combination of the payload and envelope is referred as a transport message [24].

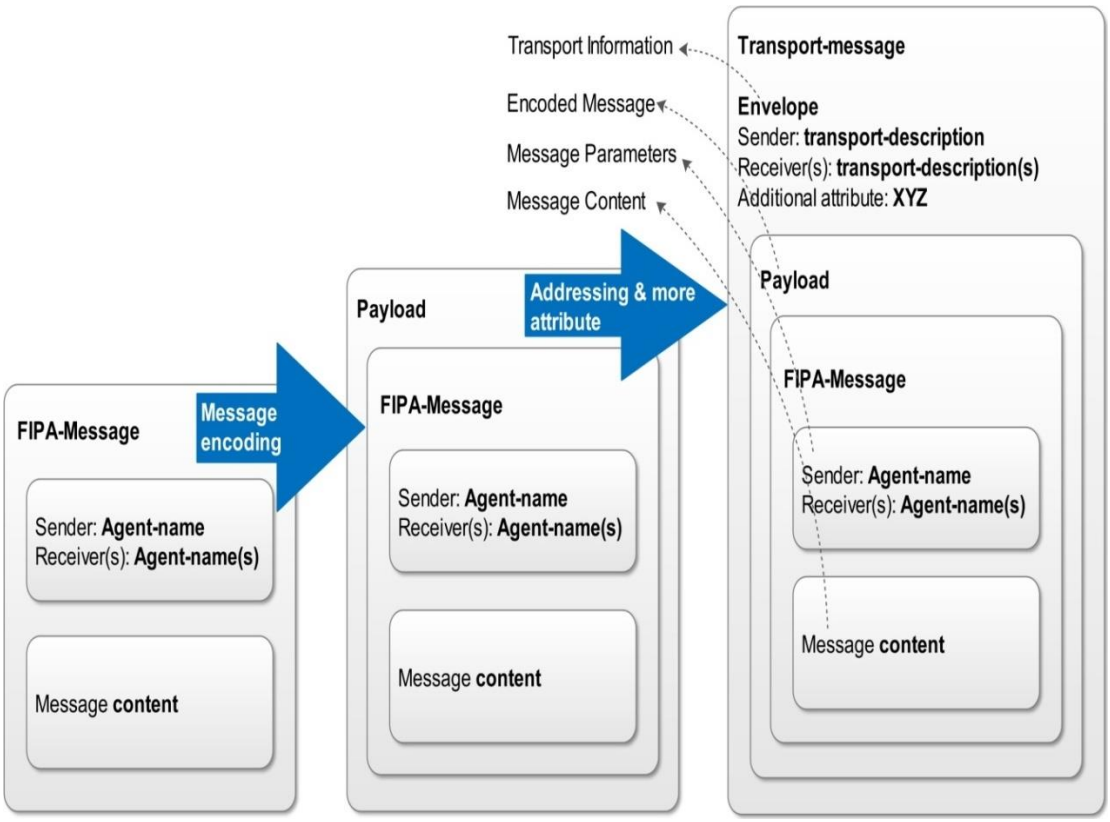


Figure 4.7 *A FIPA message to transport-message [recreated from 24].*

As shown in *figure 4.7*, a message is contained inside a transport-message when messages are sent [24]. ACL (Agent Communication Language) provides a mean for agents to share information and knowledge. ACL is needed for agents to interact with each other in a shared language, hiding their internal details and to build communities of agents that can tackle the problems collectively that no individual agent can. KQML (Knowledge Query and Manipulation Language) and FIPA ACL are fully specified existing ACLs [45]. The FIPA ACL is a standard message language that sets out the encoding, semantics and pragmatics of the messages [44]. The syntax of the FIPA ACL is similar to KQML communication language. However, there are fundamental differences between KQML and FIPA ACL despite their syntactic similarity [44]. They differ primarily in the details of their semantic frameworks. Difference between KQML and FIPA ACL is discussed in [45]. Several other means and approaches have been implemented over the years to achieve communication between agent based systems; from Remote Procedure Call (RPC) or Remote Method Invocation (RMI), to CORBA (Common Object Request Broker Architecture) [63] and Object Request Brokers (ORB's) [64]. Overall, the goal has been the same, to exchange information and knowledge between agents. According to [45], however, ACLs like KQML or FIPA ACL stands a level above than other mentioned approaches, because:

- They handle propositions, rules and actions instead of simple objects.
- The ACL message describes a desired state in a declarative language, rather than a procedure or method.

When using an ACL, agents transport messages over the network using some lower-level protocols (SMPT, TCP/IP, IIOP, HTTP, etc.) [45].

Some of the basic notions about agents and their communications can be summarized as below [24]:

- Each agent has an agent-name. This agent-name is unique and unchangeable.
- Each agent 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 IIOP, SMTP, or HTTP. A transport is a mechanism for transferring messages.
- A transport-message is a message that is sent from one agent to another in a format that is appropriate to the transport being used.

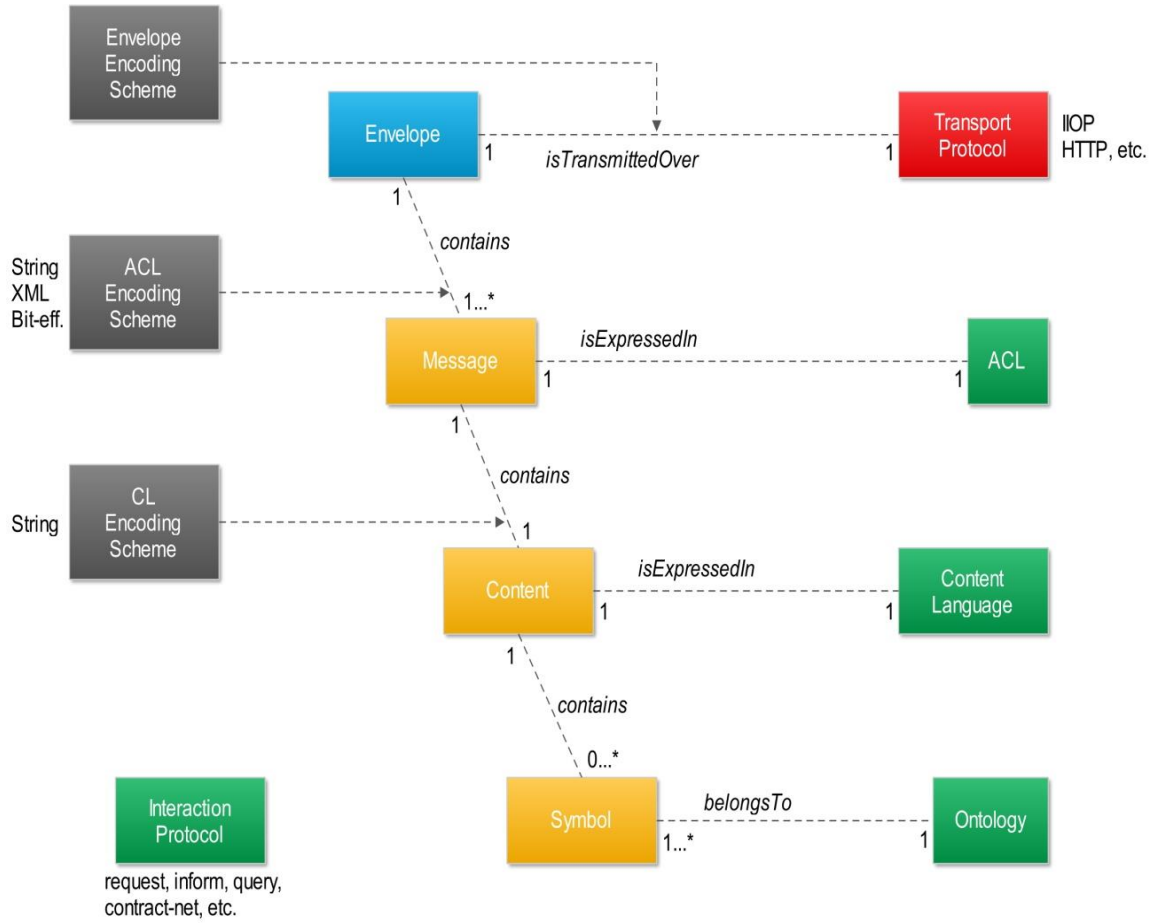


Figure 4.8 *FIPA-ACL communication model [recreated from 43].*

The *figure 4.8* represents FIPA-ACL communication model. The connection between FIPA-ACL communication model and the application layer of the classical OSI stack is shown the *figure 4.9*. The FIPA-ACL model starts on top of the OSI reference model and can be separated into several FIPA-ACL computation layers within the application layer of OSI stack [70][81].

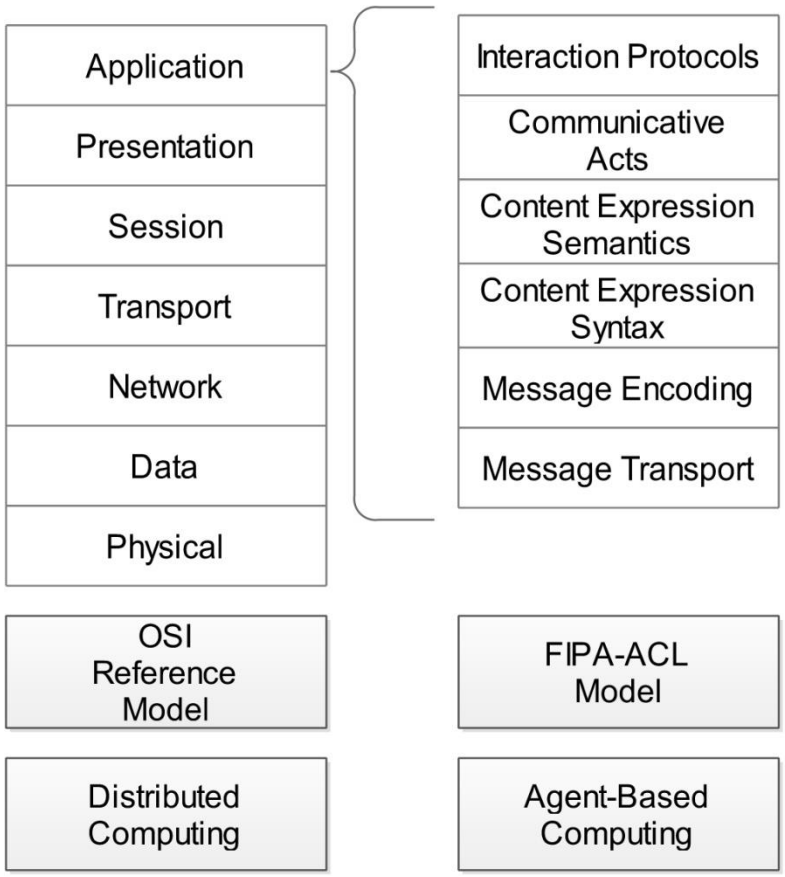


Figure 4.9 *Connection between FIPA-ACL communication model and OSI reference model [recreated from 81].*

Specifying message exchange as a protocol defines a set of rules that message must obey to be correctly formed [65]. Here, at the message transport layer in FIPA-ACL communication model, agents use some lower-level protocols, such as, SMPT, TCP/IP, IIOP, HTTP, etc. to interchange messages through a physical network. The next layer, message encoding layer validates message structure and encoding. The message encoding protocol enables message exchange to use multiple encodings such as the String encoding [66], XML encoding [67], and Bit-Efficient encoding [68]. In content expression syntax layer, agents recognize the entities of the content of messages and determine whether the content structure is correct based on common content language representation. Another layer, content expression semantics layer defines the use of ontologies to describe the meaning of the content of the message. The communicative acts layer de-

fines the intention of the sender of the message. And, the interaction protocols specify communication patterns of agents [81].

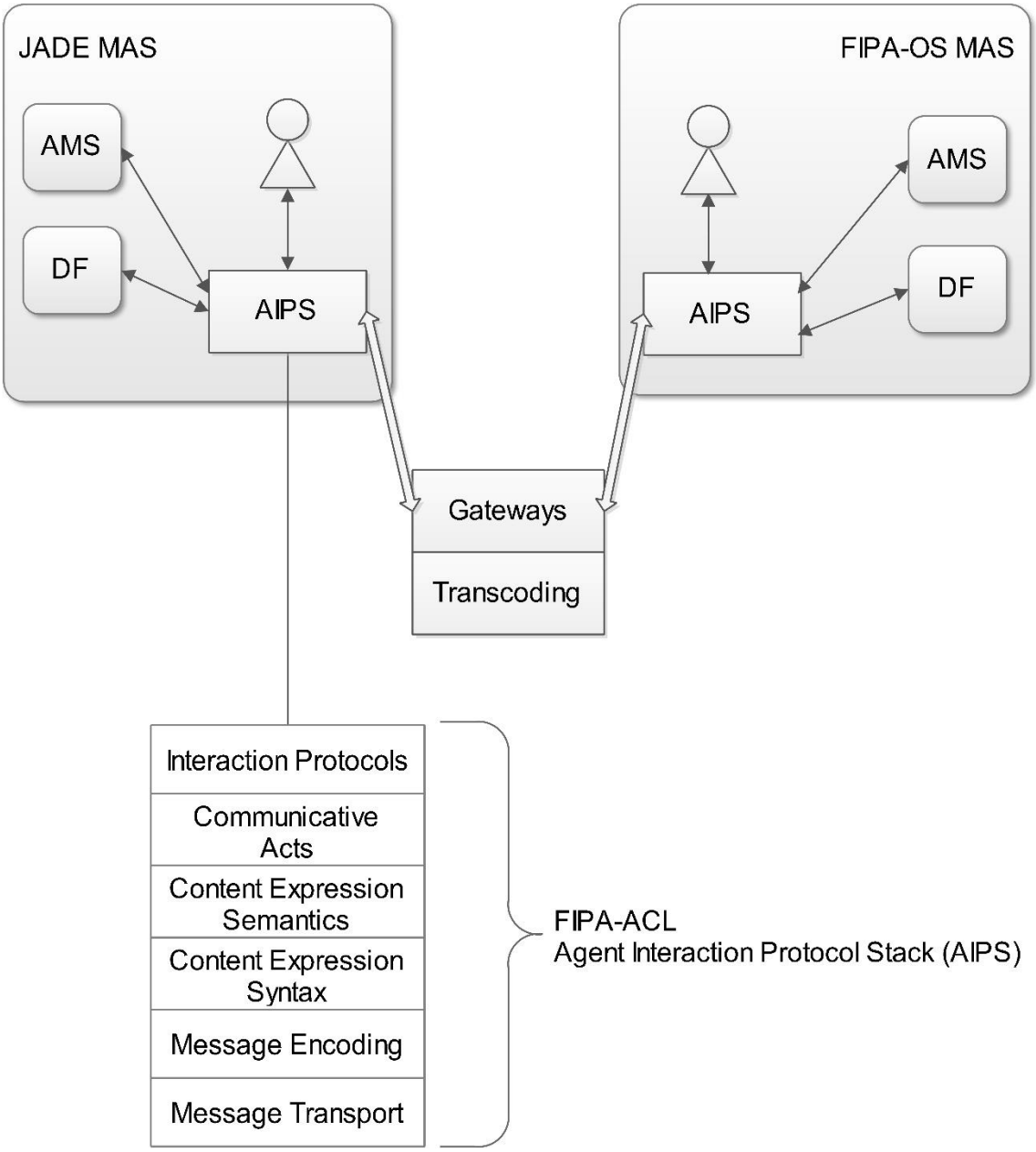


Figure 4.10 *FIPA specifies Agent Interaction Protocol Stack (AIPS) for Multi-agent System (MAS) interaction [recreated from 65].*

4.2 Agent management

In addition to agent communication, agent management is another fundamental aspect of agent systems introduced by FIPA. According to FIPA Agent Management Specification [30], agent management provides the normative framework within which FIPA

agents exist and operate. It establishes the logical reference model for the creation, registration, location, communication, migration and retirement of agents [30]. The logical components contained in the agent management reference model are depicted in *figure 4.4*.

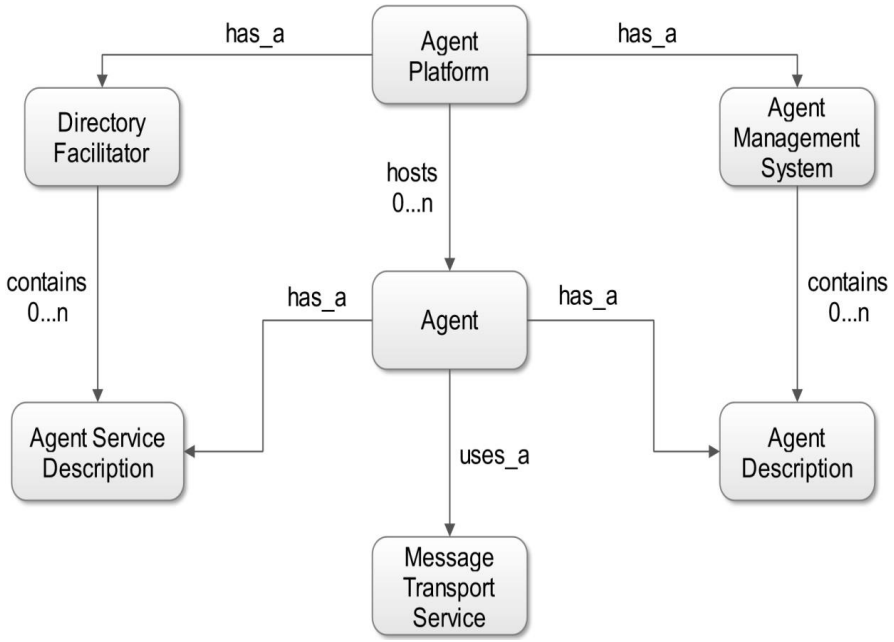


Figure 4.11 *Representation of agent management [recreated from 43].*

In an agent platform [23], an agent is a fundamental actor. Agent platform provides the physical infrastructure in which agents are deployed. It is an environment where agents act autonomously. Agent platform as a physical infrastructure consists of several components like: machine(s), operating systems(s), any additional support software, FIPA agent management components, and agents themselves [23]. The implementation details and internal design of an agent platform is left to the developers of an agent system and is not a subject of FIFA standardization. FIPA only mandates the structure and encoding of messages used to exchange information between agents and other third party FIPA compliant technologies [70].

As mentioned in [3] and [23], the FIPA agent management component consists of:

1. Agent Management System (AMS)
   The AMS [3][23][30] controls access and use of the agent platform. It is responsible for maintaining a directory of resident agents and for handling their life cycle. It provides white page services like maintaining directory of agent location, naming and control access services. Each agent must be registered with an AMS to obtain an Agent Identifier (AID) [23]. The AMS maintains the directory of all agents residing within the agent platform. An AMS is a mandatory component of the agent platform. The AMS defines the core directory actions such as regis-

ter, deregister, modify, search, and get-description [30]. In addition to the management functions exchanged between the AMS and agents on the agent platform, the AMS can instruct agent platform to perform following operations: suspend agent, terminate agent, create agent, resume agent execution, invoke agent, execute agent, and resource management [30].

2. Message Transport Service (MTS)
   The Message Transport Service is a service provided by the agent platform to which the agent is attached [2]. The MTS [23] supports transportation of FIPA Agent Communication Language messages between agents in any given agent platform or between agents residing in different agent platforms. The FIPA Agent Message Transport Specification [36] deals with the delivery and representation of messages across different network transport protocols [3]. On any given agent platform, the MTS is provided by an ACC (Agent Communication Channel) [36]. The ACC uses information provided by the Agent Management System to route messages between agents within the platform and to agents residing on other platforms. The agent message transport reference model is shown in *figure 4.12*.

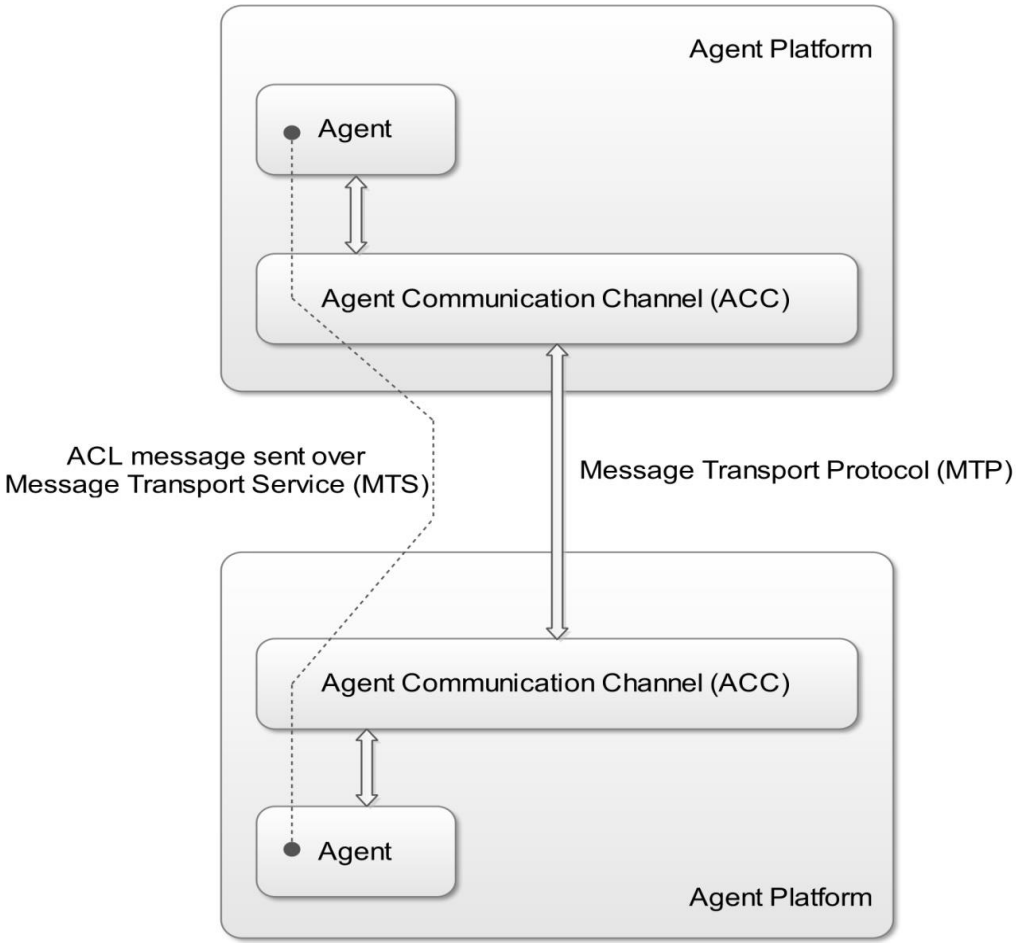


Figure 4.12 *Agent message transport reference model [recreated from 36].*

Here, the Message Transport Protocol (MTP) carries out the physical transfer of messages between two Agent Communication Channels (ACCs). The ACL (Agent Communication Language) represents the content of the messages carried by both the MTS and MTP.

3. Directory Facilitator (DF)
   The Directory Facilitator [23] provides yellow page services to other agents. It is an optional component of the AP. Agents registers their application specific services with the DF. Moreover, agents can query the DF to find out what services are offered by other agents, including the location of agents and their offered services in ad hoc networks [23].

5. MAPPING/COMPATIBILITY BETWEEN HTML5 AGENT FRAMEWORK AND FIPA

This chapter analyses compliance and compatibility to make HTML5 Agent Framework a FIPA-compliant system. Some of the aspects of HTML5 Agent Framework such as agent communication and agent management are in the initial stage of implementation and have their own architecture specific functions. Large scale realization of agent applications in modern commercial and industrial settings need compatibility and interoperability across network of agent systems. Compatibility and interoperability between heterogeneous agent systems can only be achieved by adopting one of the agent systems standards. Complying multiple agent systems with one of the agent system standards allows doing of the things in the same way. So, use of existing standard like FIPA specifications for agent system development can be a benefit, because it is easier to obtain interoperability and compatibility between heterogeneous agent systems based on existing well-known standards than on self-made standard. Therefore, in this chapter, comparisons are made between FIPA specified system and HTML5 Agent Framework. Furthermore, approaches to make HTML5 Agent Framework a FIPA-compliant system are discussed in later part of this chapter.

An agent system that conforms to FIPA specifications is termed as a FIPA-compliant system. A FIPA-compliant system has an ability to interoperate with other heterogeneous agent systems that are FIPA-compliant as well. Interoperability is fundamentally guaranteed between agent systems that are built in accordance with FIPA specifications. The conversion of an agent system/MAS into a FIPA-compliant system is one way to support interoperability between different agent systems that are FIPA-compliant [75]. Different multi-agent systems are not interoperable due to differences in their architectural elements, such as differences in syntax and semantics of agent communication language, and incompatible message transport mechanisms [85]. As pointed out in [3], [22] and [75], for an agent system to be considered a FIPA compliant, it must at least implement the FIPA “Agent Management” and “Agent Communication Language” specifications. Different implementations and design approaches are adopted when engineering agent systems. Similarly, the current implementation of HTML5 Agent framework [20] has its architecture specific agent management and agent communication model. General descriptions about agent communication and agent management model in HTML5 Agent Framework are already pointed out in *section 3.1.1* and *section 3.1.2* respectively. Some of the findings of comparison between these two systems can be summarized as below:

1. Simple agent management functionality for agents is implemented in Agent Server [14]. The latest version of HTML5 Agent Framework [20] introduces a centralized entity called Management Server that provides additional management functionality. However, future implementation requires more information about agent application, such as description of agents, the configuration of the agent, etc. as mentioned in FIPA "Agent Management Specification". The agent management functionalities specified in FIPA "Agent Management Specification" can be incorporated in current implementation to provide better agent management functionalities.

2. A yellow page service or Directory Facilitator (DF) can also be implemented for advertising and looking up agent capabilities. However, this is not a mandatory component in FIPA-compliant agent framework

3. In current implementation [20], communication model is very generic and need to be refactored to implement standard FIPA-ACL (Agent Communication Language) for agent communication. So, the communication infrastructure should support standardized FIPA-ACL to rationalize the use of the current framework in a network of other FIPA-compliant technologies.

4. The ability for an interaction/interoperation with agents in third-party FIPA-compliant multi-agent systems, such as JADE [12], Radigost [69], etc. must be introduced. Three cases of agent-to-agent communication are mentioned in HTML5 Agent Framework. In order to make HTML5 Agent Framework interoperable with third-party FIPA-compliant multi-agent systems, inter-platform communication component should be implemented using some lower-level protocols (SMPT, TCP/IP, IIOP, HTTP, etc.) as mentioned in FIPA communication model.

So, an agent system when mapped with FIPA requires that it manages the agent's life cycle, provides a communication infrastructure, and may include an agent directory interface [69]. Moreover, at a more advanced level, an agent system should incorporate various security features and there should be infrastructure for flexible agent mobility/migration (navigation model).

5.1 Proposed solution

In this section, references [3] and [75] are used as input for the design of the proposed solution to make HTML5 Agent Framework a FIPA-compliant one. An approach of converting an MAS into a FIPA-compliant system requires that developers of agent system must build their system based on FIPA specifications. For this, developers of

agent system should impose changes on the current system architecture to conform to the new standards. As described in [3] and [75], there could be two possible approaches to conform a MAS into a FIPA-compliant one:

1. Modify the whole system architecture based on the guidelines provided by FIPA specifications, i.e. for an agent platform to be considered FIPA-compliant, it must at least implement the FIPA "Agent Management" and "Agent Communication Language" specifications. That is, the approach is to convert the whole system to adhere to FIPA specifications [3][22][75].

2. Modify just a part of the system's architecture. With this, the actual architecture of the system remains the same but FIPA-compliant gateway is incorporated to the system that works as interoperability sub-system to communicate with other third-party FIPA-compliant multi-agent systems [3].

The first approach is a usual approach of converting a MAS into a FIPA-compliant system. For this, developers of agent system should rebuild the whole system to adhere to FIPA specifications. This requires extensive code rewriting, re-design and testing, and may or could not be considered radical approach. Based on this approach, an attempt to make HMTL5 Agent Framework a FIPA-compliant system requires modifications based on the guidelines provided by FIPA specifications. This requires changes in current implementation of agent management and agent communication model in HTML5 Agent Framework.

As already mentioned, the alternative approach of converting a MAS into a FIPA-compliant will be to implement FIPA-compliant gateway component in existing agent system (non FIPA-compliant). Referring [4], "a gateway is an element of an agent system that is capable of mediating some form of interoperability with another agent system". The concept behind gateway is also discussed in FIPA Abstract Architecture Specification [24]. According to [24], "where direct end-to-end interoperability is impossible, impractical or undesirable, it is important that consideration be given to the specification of gateways that can provide full or limited interoperability". Services offered by a gateway can include following [4] [24]:

- A gateway may provide message transfer between two agents that use different transports or message encoding representation.
- A gateway may support agent service advertisement and discovery by translating between different directory services.

So, gateway as a component of a non FIPA-compliant system provides a medium for it to interact with agents hosted in third-party FIPA-compliant MASs. For instance, an agent located in JADE [12], Radigost [69], etc. can directly interact with HTML5 Agent Framework agents by sending a message through the gateway component, and vice-

versa. However, there needs to be a way for a message to travel across the system boundaries and reach the external MAS in a recognizable format, and vice-versa [69]. Most importantly, to integrate gateway component within the existing non-FIPA compliant agent system to support interoperability with agents hosted in separate agent environment will be the major challenge.

A gateway, according to [3] and [75] is the FIPA-compliant part of the system which has all of the mandatory, normative components of the FIPA architecture. Adoption of the FIPA-compliant gateway by the SARA (Synthetic Aperture Radar Atlas) multi-agent system is discussed in [3] and [75]. As described in [3] and [75], the FIPA gateway has all the normative components of the FIPA architecture as defined by FIPA specifications. The gateway agent communicates with agents in external FIPA-compliant MAS using the FIPA-ACL. It translates the incoming messages from external FIPA-compliant MAS to a form understood by its internal agents. Likewise, it translates the internal agents' request into FIPA-ACL messages, in order to be understood by external FIPA-compliant MAS. The gateway agent contains a list of the agents in the system along with their registered service in DF (Directory Facilitator). So, based on service requested by the external FIPA-compliant MAS agents, the gateway agent knows to which agent the message should be forwarded after translating the message into the form understood by that appropriate agent. The gateway agent acts as an interface between existing agent system and external FIPA-compliant MAS.

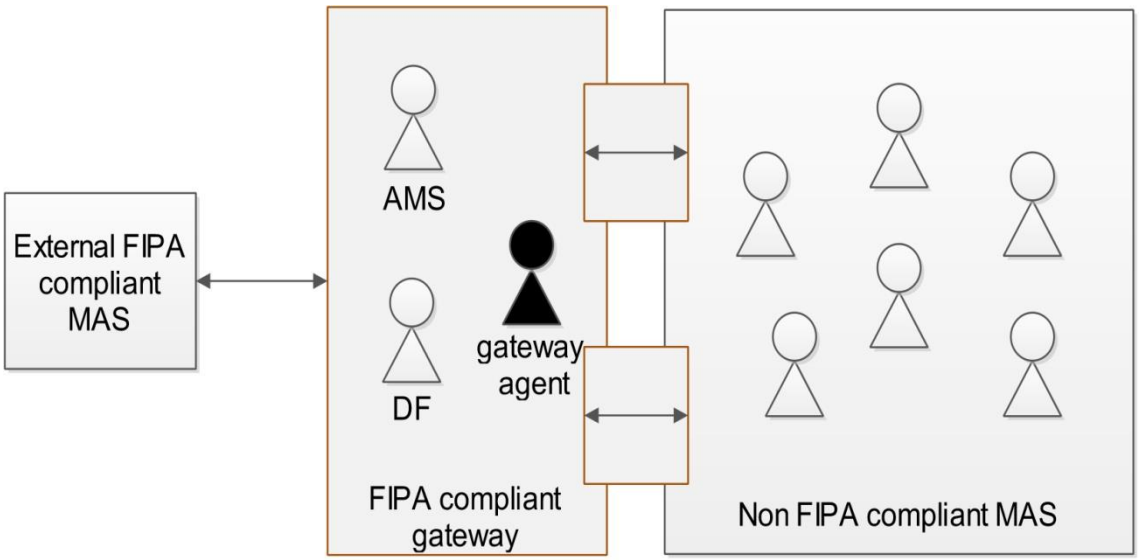


Figure 5.1 *FIPA-compliant gateway approach to make an agent platform FIPA-compliant [recreated from 3 and 75].*

With this approach, interoperability can be achieved with the use of the FIPA-compliant gateway without actually affecting the actual agent system. That means, it is not neces-

sary to conform the whole system based on FIPA guidelines [75]. Some of the advantages of adopting the FIPA-compliant gateway in existing MAS are also discussed in [3] and [75], which includes following advantages:

1. **System's architecture remains the same as before**, implementation is only needed for the FIPA-compliant gateway and the interaction between gateway agents with the other agents of the current agent system. Consequently, developers can avoid the complexity of amending the whole system. They can just work only in a specific part of the system, i.e. the FIPA compliant gateway [3] [75].

2. **Increased security**, isolating the interoperable part of the system, i.e. gateway from the rest of the system increases security. The internal property of the current system remains hidden from external third-party MAS due to the FIPA-compliant gateway. The interaction between the system agents and external MAS is managed by the gateway agent; this protects the system's agents (for instance, HTML5 Agent Framework), hardware/software resources from being accessed by external components. So, the FIPA-compliant gateway can act as a shield for the core system. Securing the FIPA-compliant gateway implies minimum security for the rest of the system. The more secure the FIPA-compliant gateway, the less security is needed for the rest of the system [3] [75].

As already pointed out, FIPA specifications exist to support interoperability across heterogeneous agent systems that are FIPA-compliant as well. As described in [75], the conversion of an agent system which does not need to communicate with external agent systems into a FIPA-compliant system would be useless, since the agents that belong to the particular agent system are identical and can obviously interoperate between themselves [75]. In other words, if agents share common architecture, then they can communicate with each other without any problems. For instance, in HTML5 Agent Framework as well, a simple agent-to-agent communication framework is implemented. This framework allows agents residing within that framework to send and receive messages between themselves regardless of their location [20]. The existence of FIPA agent management components: Agent Management System (AMS), Message Transport Service (MTS), and Directory Facilitator (DF), which are mandatory, normative components of the FIPA specified system impose extra complexity in an agent system constituted by identical agents capable of interoperating between themselves [75]. So, the application field or user group of a non-FIPA compliant system should be realized before converting a non-FIPA system into a FIPA-compliant system, otherwise, it would be more or less useless to incorporate interesting technical aspects and possibilities.

6. PROOF OF CONCEPT/EXPERIMENTATION

Based on purposed solutions in *section 5.1*, a decision was made to design and implement FIPA-compliant gateway component for HTML5 Agent Framework and establish simple communication between FIPA-compliant JADE multi-agent system and HTML5 Agent Framework as a proof of concept for this thesis. Major challenges for this experimentation were to determine:

- How does the communication work between these two heterogeneous agent systems? The messaging protocol needs to be developed that will allow exchanging the information between these two systems.
- How message format conversion can be implemented in gateway component in HTML5 Agent Framework. Since, JADE is Java based FIPA-compliant system, there should be some mechanism to convert FIPA-ACL message from JADE to message understood by HTML5 agent in HTML5 Agent Framework and vice-versa.

The experimentation in this thesis demonstrates a simple message exchange between HTML5 Agent Framework and JADE multi-agent system. The functionalities of HTML5 Agent Framework are implemented using Web protocols and technologies [20]. The Web support (HTTP) here in JADE is added through *JadeGateway* [78] using Java Servlet [77]. *Figure 6.1* represents the outline of communication between HTML5 Agent Framework and JADE multi-agent system. Message exchange between a HTML5 agent and a JADE agent takes place via WebSocket [80]. The JADE Web application shown in *figure 6.1* represents an approach of accessing a JADE multi-agent system from a Servlet using the *JadeGateway* class. This Web application displays a button and if we click on it a send message action is invoked and a message will be sent to a JADE agent and this agent will send a reply. In this experimentation, a message from a HTML5 agent running in a Web browser in HTML5 Agent Framework is sent over HTTP via WebSocket that invokes send message action in JADE Web application. In *GatewayAgent*, the incoming message string from HTML5 agent is treated as ACL message. A legal ACL message is created in *GatewayAgent* which constitutes three parameters: *communicative act* (also called *performative*), *receiver*, and *content*. The WebSocket in *figure 6.1* serves as a channel for message flow between HTML5 Agent Framework and JADE. It provides full-duplex communication, not only a HTML5 agent can send a message but also can receive a message from JADE via WebSocket. The overall proof of concept will be described briefly in later sections of this chapter.

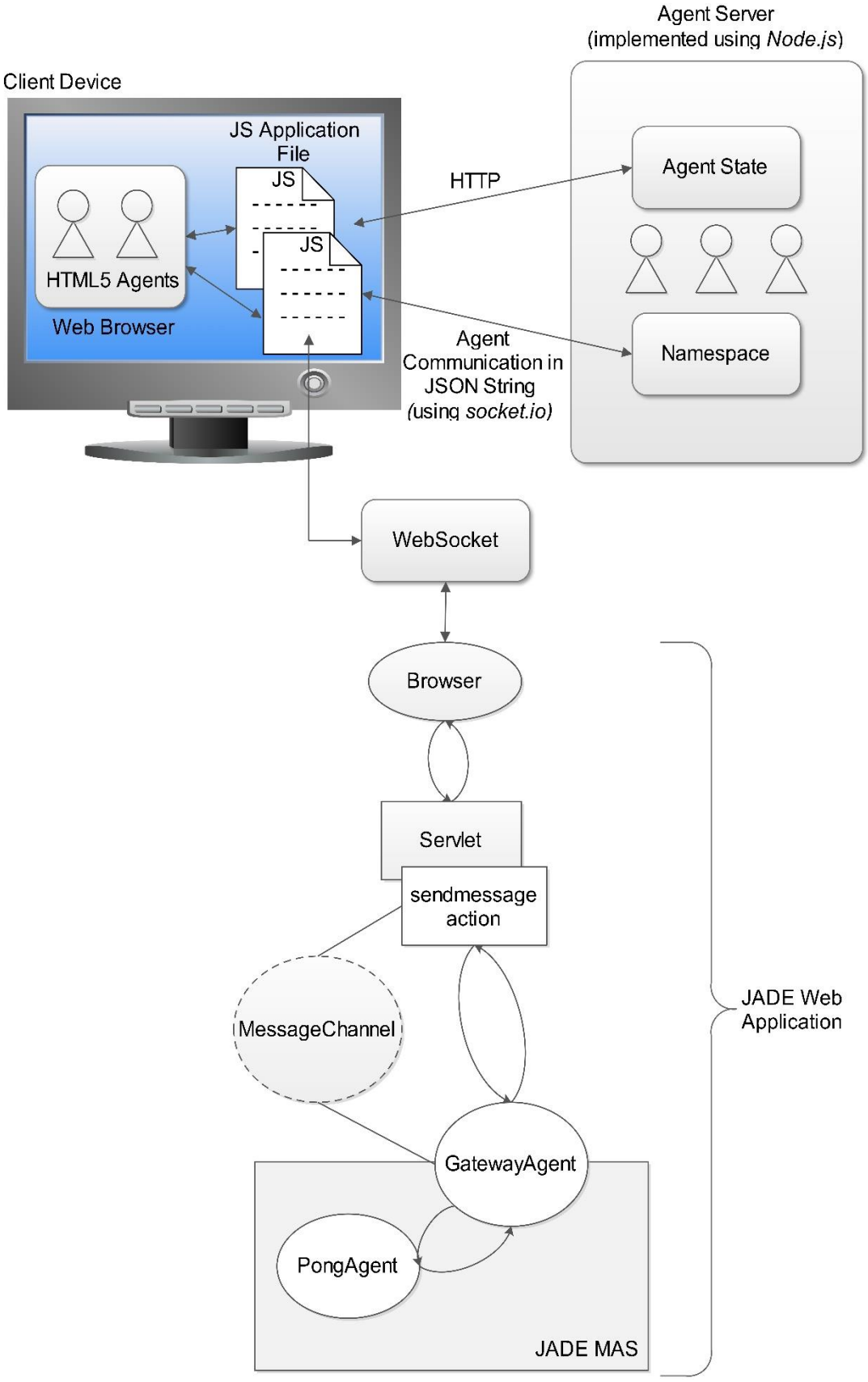


Figure 6.1 *Outline of communication between HTML5 Agent Framework and JADE.*

6.1 Implementation in JADE MAS

All the architectural details and introduction about JADE multi-agent system can be found in [12]. The approach implemented in this thesis to access JADE system is based on reference [78]. Based on [78] by Viktor Kelemen, a simple web application is created using Java Servlet [77]. JAVA Servlets are programs that run on a Web or application server and act as a middle layer between requests coming from a Web browser or other HTTP client on the HTTP server [77]. This simple web application demonstrates how a JADE agent in JADE multi-agent system can be accessed from a Servlet using the *JadeGateway*. Since, JADE has been implemented in Java, it is easy to integrate JADE and Servlet to create a Web application. In JADE, the main package we use to communicate with Servlet is *jade.wrapper.gateway*, which includes classes: *JadeGateway* and *GatewayAgent*. These classes enable a non-JADE application to issue commands to a JADE based application.

- **JadeGateway,** this class provides a simple gateway that allows some non-JADE code to communicate with JADE agents.
- **GatewayAgent,** this agent is the gateway able to execute all commands requests received via *JadeGateway*. The agent acting as a gateway will be initiated by the class *JadeGateway* from the servlet, by the method *JadeGateway.init ()*.

Here, the *GatewayAgent* serves as a gateway component in JADE system. This class can create a legal ACL (Agent Communication Language) message from incoming message string from a Web client. This property of *GatewayAgent* was further analysed to determine if it can be used as a gateway component between HTML5 Agent Framework and JADE. Based on the property of *GatewayAgent* in JADE, communication between HTML5 agent and JADE agent was tested and it worked. The ability of *GatewayAgent* to serve as a gateway component between HTML5 Agent Framework and JADE changed the dimension of the initial decision to implement FIPA-compliant gateway component for HTML5 Agent Framework. The proof of concept is different from the initial decision to implement FIPA-compliant gateway component for HTML5 Agent Framework. Here, the *GatewayAgent* in JADE is capable of mediating some form interoperability with HTML5 Agent Framework. However, it was considered as a reasonable option for this Master's thesis.

The implementation details are shown in *figure 6.2*. For this, the NetBeans IDE is used and the JADE Web application is hosted using the tomcat server [79].

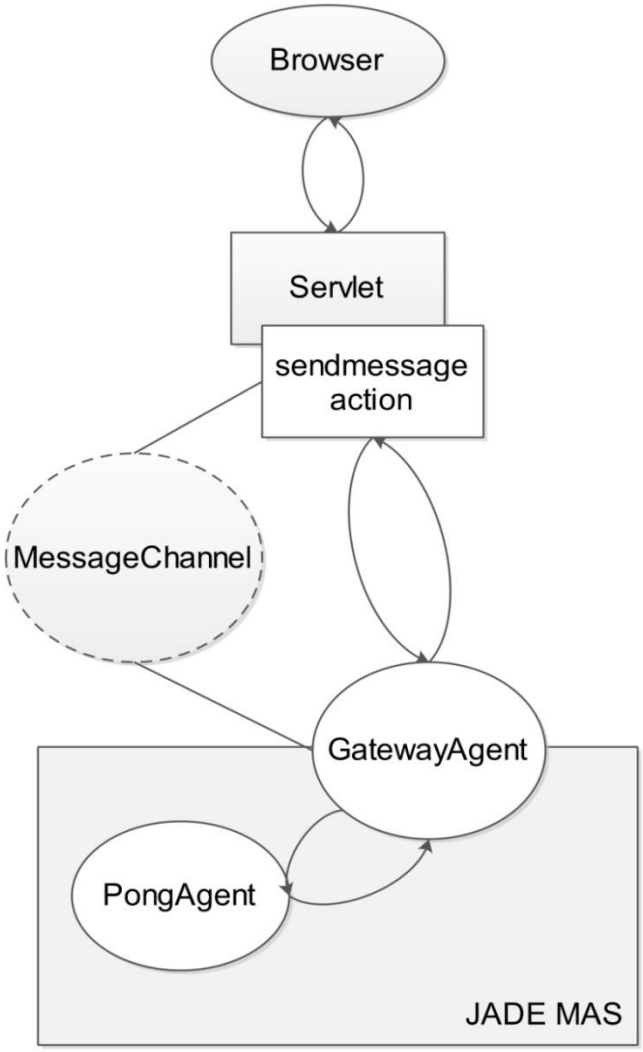


Figure 6.2 *Implementation detail for creating web application in JADE MAS [recreated from 78].*

The communication implemented between HTML5 Agent Framework and JADE constitutes following steps:

1. In the browser, HTML5 agent generates a POST message through WebSocket.
2. The Servlet handles it and the "sendmessage" action is invoked. The Servlet runs in an application server (tomcat).
3. The "sendmessage" action creates a new *MessageChannel* object which will be the message channel between the *GatewayAgent* and the Servlet. *MessageChannel* is an object created by the Servlet and used as a communication channel.
4. The *GatewayAgent* gets this *MessageChannel* object created previously and extracts the recipient and content of the message. After that, the *GatewayAgent* sends the received message from the Web client to the recipient agent (in this case, PongAgent).
5. *PongAgent* receives the message and responses to the *GatewayAgent*.

6. The *GatewayAgent* packs the reply from recipient agent and sends it via *MessageChannel* to the Servlet.
7. The Servlet forwards it to the browser.

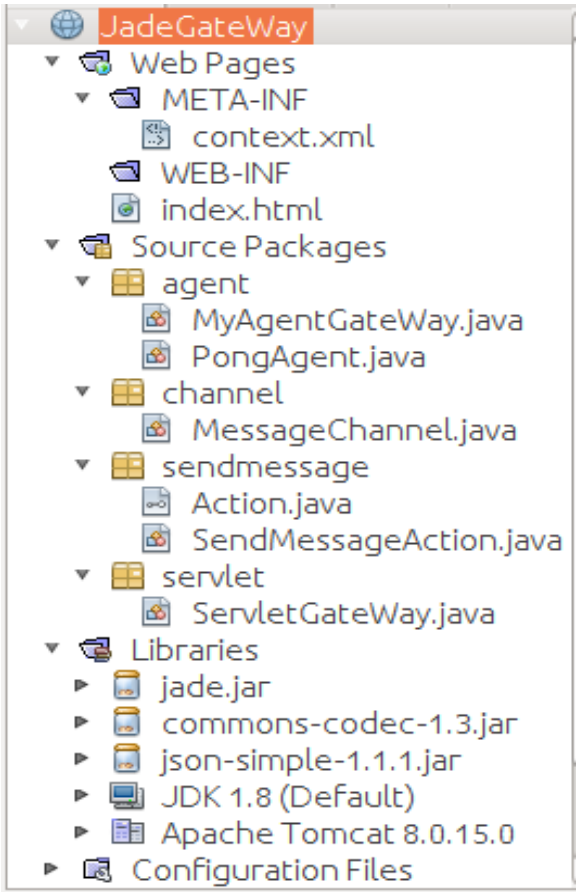


Figure 6.3 *Project structure of implementation in JADE in NetBeans IDE.*

Figure 6.3 represents overall project structure of implementation in NetBeans IDE. For this implementation, JADE library (jade.jar) is imported to be able to use some utility classes provided by JADE library. Details about project structure are described below:

HTML page (index.html)

This is a simple HTML page which contains a form where the "Send Message" button appears. It contains a hidden field which indicates the type of the action sent to the Servlet. The message from HTML5 agent invokes "sendmessage" action in this HTML page.

Listing 6.1: Snippet of “index.html” file containing form where “Send Message” button appears.

```
<body id="page">
 <br><h1>Simple JadeGateWay Example</h1><br>
 <HR>
 <form id="operationBox" method="get" action="ServletGateWay">
     <input type="hidden" name="action" value="sendmessage"/>
          <input type="submit" value="Send Message"></input>
 </form>
</body>
```

servlet package

The Servlet receives a *post* message from previously created HTML page and sends back the proper response. This JAVA package contains “ServletGateWay.java” class. “ServletGateWay.java” is responsible for processing the action to perform and deliver a response that will come from JADE.

ServletGateWay.java

The agent acting as a gateway will be initiated by the class *JadeGateway* from the Servlet by the method *JadeGateway.init ( )*. This method will receive as parameters the agent acting as a gateway (the name of the class that implements the agent), and the port number where we are running the JADE platform we want to communicate. In this experimentation, *MyAgentGateWay* class is derived as a *GatewayAgent* in the first parameter of *JadeGateway.init ( )* method.

```
// JadeGateway sets which class will be the GatewayAgent
JadeGateway.init("agent.MyAgentGateWay",null);
```

The second parameter is null because the main container of the JADE multi-agent system is running on the same host and on the default 1099 port. Here, the *get* request from html page is treated as *post* message. The *doPost* function is shown in *listing 6.2*:

Listing 6.2: Snippet of *doPost* function.

```
// We get the value of the hidden field action

String actionName = request.getParameter("action");

if (actionName == null)    {
     response.sendError(HttpServletResponse.SC_NOT_ACCEPTABLE);
     return;
}

// We make the object that implements the action interface
// In this case "action" is the "sendmessage"

Action action = (Action) actions.get(actionName);
if (action == null)    {
     response.sendError(HttpServletResponse.SC_NOT_IMPLEMENTED);
     return;
}

// SendMessageAction is performed
action.perform(this, request, response);
```

sendmessage package

This JAVA package contains "Action.java" and "SendMessageAction.java" classes. Servlet uses these classes.

Action.java

The "Action.java" acts as a generic interface for all possible actions. The Servlet defines a *perform* method so that all actions would be invoked as *action.perform (...)* as shown in *listing 6.2*.

Listing 6.3: Snippet of "Action.java" class.

```
public interface Action {

     public void perform(HttpServlet servlet, HttpServletRe-
     quest request, HttpServletResponse response)

         throws IOException, ServletException;

}
```

SendMessageAction.java

The "SendMessageAction.java" class constitutes three important steps:

1. Creating a *MessageChannel* object.
2. Filling the *MessageChannel* object with the proper content (receiver, message).
3. Sending the *MessageChannel* object to the *GatewayAgent* with the *JadeGateway.execute* method with a parameter which is the *MessageChannel* object.

Listing 6.4: Snippet of "SendMessageAction.java" class.

```
//We create a message with a recipient and a content
//"message" that we receive from HTML5 agent framework

// Creates a MessageChannel object "channel"
MessageChannel channel = new MessageChannel();

String str = request.getParameter("message");
channel.setReceiver("PongAgent");
channel.setMessage(str);

try    {

//We access JADE via JadeGateway and wait for the answer
//JadeGateway.execute sends this object "channel" to Gate-
wayAgent

     JadeGateway.execute(channel);

} catch(Exception e) { e.printStackTrace(); }
```

agent package

This JAVA package contains "MyAgentGateWay.java" and "PongAgent.java" classes.

MyAgentGateWay.java

"MyAgentGateWay.java" acts as a medium for entering to the JADE multi-agent system. It extends the *GatewayAgent* class and serves as a gateway between the Servlet and the JADE. It is also responsible for sending a message to *PongAgent* and sending back its reply to the Servlet. "MyAgentGateWay.java" gets previously created *MessageChannel* object. The *processCommand* function is invoked when *JadeGateway.execute (channel)* method is invoked. The *processCommand* is called each time a request to process a command is received. It sets the communication, receiving an object parameter that contains the information needed to perform the necessary operations. Here, an ACL message is created and sent to the recipient agent "PongAgent.java".

Listing 6.5: Snippet of "processCommand" function in "MyAgentGateWay.java".

```
MessageChannel channel = null;

// The processCommand will be invoked when JadeGateway.execute
//(object) is invoked in the servlet

@Override
protected void processCommand(java.lang.Object obj) {

  if (obj instanceof MessageChannel)    {

     channel = (MessageChannel)obj;

     // ACL message is created and sent

     ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
     msg.addReceiver(new AID( channel.getReceiver(),
     AID.ISLOCALNAME) );
     msg.setContent(channel.getMessage());
     send(msg);
  }
```

In the background *CyclicBehaviour* waits for the response and *releaseCommand* is invoked when response is received from replying agent. The *releaseCommand* method notifies that the command has been processed and remove the command from the queue.

Listing 6.6: Snippet of "releaseCommand" method in "MyAgentGateWay.java".

```
// In the background CyclicBehaviour waits for the response.
// releaseCommand will be invoked when response is received
// from replying agent(PongAgent)

addBehaviour(new CyclicBehaviour(this)
{
    @Override
    public void action() {

       ACLMessage msg = receive();

       if ((msg!=null)&&(channel!=null))    {
           channel.setMessage(msg.getContent());

           // The response is returned in the channel object
           // to the Servlet

           releaseCommand(channel);
       } else block();
    }
});
```

PongAgent.java

"PongAgent.java" class creates an agent that runs inside JADE multi-agent system. This is the agent with which JADE web interface tries to communicate. Here, the content of reply message is defined.

Listing 6.7: Snippet of "PongAgent.java".

```
// Waits for the message
addBehaviour(new CyclicBehaviour(this)
{
    @Override
    public void action(){

        ACLMessage msg = receive();
        String content= "";

        if (msg!=null) {

          // Fill the contents of the reply

          content=
          "<br/> - "+myAgent.getLocalName() + " received: " +
          msg.getContent()+
          "<br/> - "+myAgent.getLocalName() + " sent: " + "I
          got the  time";

          ACLMessage reply = msg.createReply();
          reply.setPerformative( ACLMessage.INFORM );
          reply.setContent(content);
          send(reply);
          System.out.print(content);

        }else block();
   }
});
```

Back to the Servlet

The message content from "PongAgent.java" is returned to the *CyclicBehaviour* in "MyAgentGateWay.java". Inside *CyclicBehaviour*, *releaseCommand* is invoked as shown in *listing 6.6* and the response is returned to the Servlet in *MessageChannel* object: *releaseCommand (channel).* It creates a simple JSON output that contains proper response. For this, "json-simple-1.1.1.jar" is imported in NetBeans IDE. "JSON.simple" is a simple JAVA toolkit for JAVA. It is used to encode or decode JSON text.

Listing 6.8: Snippet of reply from JADE agent in “SendMessageAction.java”

```
try    {

   // We access JADE via JadeGateway and wait for the answer
   // JadeGateway.execute sends this object channel to
   // GatwayAgent

   JadeGateway.execute(channel);

} catch(Exception e) { e.printStackTrace(); }

// Creates output

response.setContentType("application/json");
PrintWriter out = response.getWriter();

// We print the reply received from JADE in JSON format

JSONObject obj = new JSONObject();

obj.put("name", "Message has been sent");
obj.put("reply", "Reply:"+channel.getMessage());

out.print(obj);
out.flush();
out.close();
```

Figure 6.4 represents important steps that take place in execution in JADE.

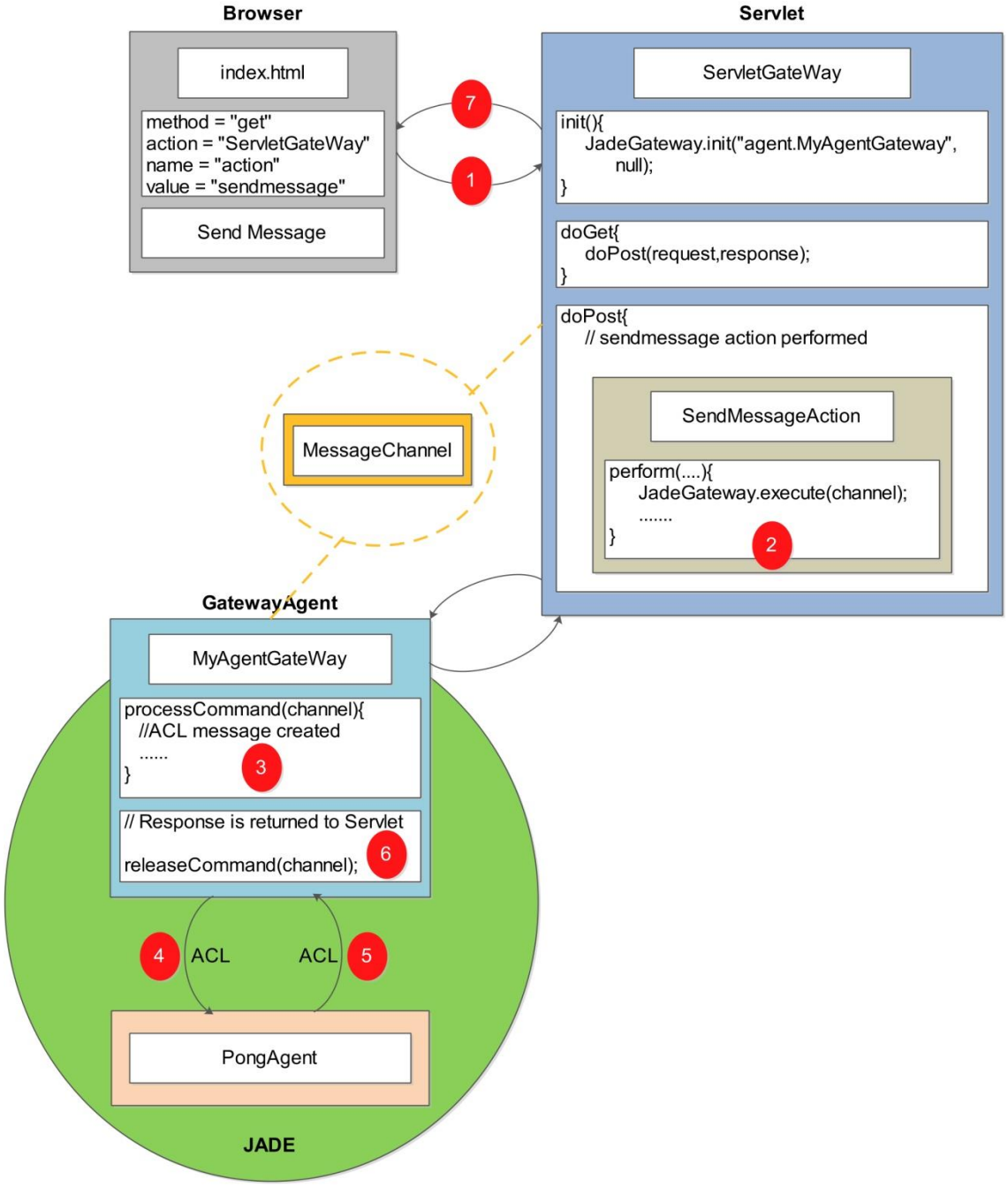


Figure 6.4 *Steps in execution in JADE.*

Starting JADE

At first, main-container of JADE multi-agent system is launched. The following command starts the main-container and a GUI interface of JADE multi-agent system.

```
sks@sks-Inspiron-N4010:~/NetBeansProjects/JadeGateWay$ java -classpath .:/home/sks/NetBeansProjects/JadeGateWay/lib/jade.jar jade.Boot -gui
```

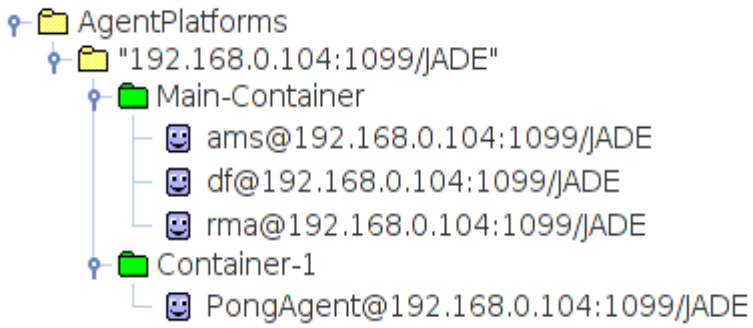
After that, “PongAgent.java” class is compiled which contains code for an agent called PongAgent.

```
sks@sks-Inspiron-N4010:~/NetBeansProjects/JadeGateWay$ javac -classpath lib/jade.jar -d classes/ src/java/agent/PongAgent.java
```

Then, the agent PongAgent is started in JADE container.

```
sks@sks-Inspiron-N4010:~/NetBeansProjects/JadeGateWay$ java -classpath .:/home/sks/NetBeansProjects/JadeGateWay/lib/jade.jar:classes/ jade.Boot
-container PongAgent:agent.PongAgent
```

After successfully executing these commands in terminal, a JADE GUI appears.



6.2 Implementation in HTML5 agent framework

The implementation in HTML5 Agent Framework constitutes usage of WebSocket [80]. As already pointed out, the WebSocket serves as a channel for message flow between HTML5 Agent Framework and JADE. The code snippet of “websocket.js” is shown in *listing 6.9*. An agent named “ClockExample” created by [14] is used as a test agent in HTML5 Agent Framework. This agent increases a variable “index” in regular interval and the value of that variable is displayed as time in “ClockExample.html” in a Web browser. A “Send Message” button is created in “ClockExample.html”, and the *send ( )* function is invoked when “Send Message” button is clicked. This *send ( )* function captures a value of variable “index” of “ClockExample” HTML5 agent and sends it as a message string to JADE via WebSocket. The code snippet of *send ( )* function is shown in *listing 6.10*. The HTTP(S) call is made to JADE Web application via WebSocket. This HTTP(S) call triggers the “sendmessage” action in JADE Web application. The Servlet in JADE application handles this HTTP(S) call and JADE is accessed using *JadeGateway* class. When a JADE agent (PongAgent) receives the message, it acknowledges the sender with a reply message (for instance, “I got the time” with INFORM *performative*). The reply message constitutes two ACL (Agent Communication Language) message parameters: *communicative act* (also called *performative*), and *content*. This reply message is returned to Servlet and creates a JSON output in a browser in JADE Web application. While the WebSocket is still open, this JSON output is parsed and displayed in the Web browser in HTML5 Agent Framework as shown in *listing*

6.10. The representation of message flow between HTML5 agent and JADE agent is shown in *figure 6.5*.

Listing 6.9: Code snippet of “websocket.js”.

```
socket.on('request', function(request) {
   var connection = request.accept(null, request.origin);

   connection.on('message', function(message) {
      console.log(message.utf8Data);

      // Load the request module
      var requests = require('request');

      // Configure the request
      var options = {
       url:'http://localhost:8084/JadeGateWay/ServletGateWay',
       method: 'POST',
       form: {'action': 'sendmessage', 'message': mesage.utf8Data}
      }
      console.log(options);

      requests(options, function (error, response, body) {
            if (!error && response.statusCode == 200) {
                  connection.sendUTF(body);
            }else{
                  console.log(error);
            }
      });

   });
});
```

Listing 6.10: Code snippet of send ( ) function in “send.js”.

```
//Once the websocket has been started, the code below is used
//to establish connection to the websocket

//This function sends the content of "label id='second'" in
//ClockExample.html in variable "sendMessage"

function send() {
 var socket = new WebSocket('ws://localhost:1337');
 var sendMessage = document.getElementById('second').innerHTML;

 socket.onopen = function () {
      socket.send(sendMessage);
 };

//Displays the received message from JADE in div ID "content" in
//ClockExample.html.
 socket.onmessage = function (message) {
   var mydata = JSON.parse(message.data);
   document.getElementById('content').innerHTML = myda
ta.name+"<br/>"+mydata.reply;
};
```

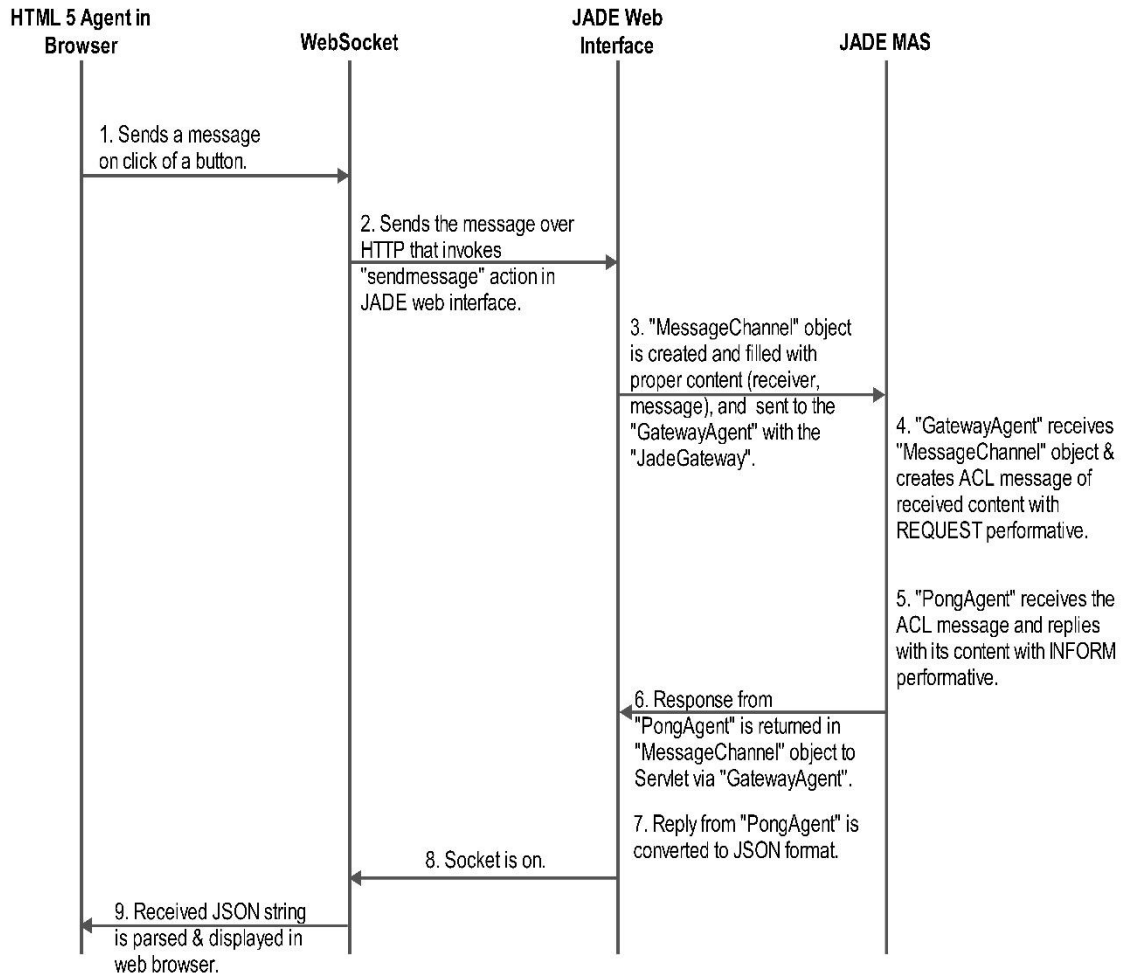


Figure 6.5 *Representation of message flow between HTML5 agent and JADE agent.*

7. EVALUATION

The approach implemented in this thesis to establish communication between HTML5 Agent Framework and JADE utilizes a library of classes offered in JADE. The WebSocket program is written in HTML5 Agent Framework side which acts as a channel for message flow between two different systems. The conception (Servlet – GatewayAgent – JADE MAS) based on [78] is one of the possible ways to access JADE from external agent system. There are also other alternative approaches to access JADE. Because, the *GatewayAgent* class in JADE can treat a simple message string as a FIPA-ACL message, this class is used to provide some form of interactivity between HTML5 Agent Framework and JADE. The proof of concept implemented in this thesis is limited or incomplete. It does not make HTML5 Agent Framework a FIPA-compliant system but has some compatibility with JADE system. The HTML5 Agent Framework relies on the functionalities that are defined in another FIPA-compliant JADE system. It is more practical to implement FIPA-compliant gateway in HTML5 Agent Framework to support interoperability and compatibility with any third-party FIPA-compliant systems. Implementing gateway component in existing non-FIPA compliant system helps defining an interface in our own system that can be implemented for each supported third-party FIPA-compliant systems.

8. CONCLUSIONS

This thesis presented an introduction to FIPA and its specifications. The purpose of this thesis was to analyze compliance and compatibility of HTML5 Agent Framework to make it FIPA-compliant system. For this, architecture and implementation specific mapping was done between HTML5 Agent Framework and FIPA agent management reference model.

The HTML5 Agent Framework developed in TUT has its own framework specific functionalities. For instance, it has its own implementation for agent management and agent communication model. When compared with FIPA reference model, it was realized that the current implementation of HTML5 Agent Framework should undergo radical changes to become FIPA-compliant. The agent management and agent communication model in HTML5 Agent Framework are very simple and are in initial stage of their implementation. The functionalities defined in FIPA "Agent Management Specification" can be incorporated in current implementation of HTML5 Agent Framework to make it FIPA-compliant. Moreover, the agent communication model in HTML5 Agent Framework should be re-designed to implement standard FIPA-ACL specification for communication. Incorporating all the components and specifications defined in FIPA reference model would require extensive code re-writing and re-design, which is considered usual approach of conforming HMTL5 Agent Framework or any other non-FIPA compliant system into a FIPA-compliant system. In addition, the alternative approach of using FIPA-compliant gateway component for converting a non-FIPA compliant system into a FIPA-compliant is discussed in this thesis. With this, the system's architecture remains the same as before and developers can avoid the complexity of amending the whole system based on FIPA reference model. Implementation is only needed for the FIPA-compliant gateway. The gateway should also adhere to those FIPA specifications.

FIPA exists to support interoperability and compatibility between heterogeneous FIPA-compliant agent systems. Engineering agent systems based on agent standards like FIPA provides way to do things in the same way. The differences among agent systems prevent interoperability and large scale realization of agent applications in modern commercial and industrial settings.

REFERENCES

[1] FIPA, the Foundation for Intelligent Physical Agents, FIPA Official Site: http://www.fipa.org/, last visited: 27.04.2015.

[2] FIPA 97 Specification, Version 2.0, Part 2, Agent Communication Language, FIPA Official Site: http://www.fipa.org/specs/fipa00003/OC00003A.html, last visited: 27.04.2015.

[3] C. Georgousopoulos, O. F. Rana, "An approach to conforming a MAS into a FIPA-compliant system", In Proceedings of First International Joint Conference on Autonomous Agents and Multi-Agent Systems, part 2, July 2002, Bologna, Italy, pp. 1-3.

[4] L. Heimo, H. Heikki, "Software Agent Technology, FIPA Agent Framework". Available at: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=A7343ACE8C17525872C2ABD47E089334?doi=10.1.1.200.5027&rep=rep1&type=pdf, last visited: 29.04.2015.

[5] FIPA agent communication specification, FIPA Official Site: http://www.fipa.org/repository/aclspecs.html, last visited: 28.04.2015.

[6] FIPA ACL message structure specification, FIPA Official Site: http://www.fipa.org/specs/fipa00061/SC00061G.html, last visited: 28.04.2015.

[7] Publicly available agent platform implementations, FIPA Official Site: http://fipa.org/resources/livesystems.html, last visited: 28.04.2015.

[8] General Public License (GNU), http://en.wikipedia.org/wiki/GNU_General_Public_License, last visited: 29.04.2015.

[9] Eclipse Public License (ECL), http://en.wikipedia.org/wiki/Eclipse_Public_License, last visited: 29.04.2015.

[10] N. Howden, R. Ronnuist, A. Hodgson, A. Lucas, "JACK Intelligent Agents – Summary of an Agent Infrastructure", Agent Oriented Software Pty. Ltd, Australia, pp. 1-2.

[11] Lesser Genera Public License (LGPL), http://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License, last visited: 29.04.2015.

[12] JAVA Agent Development Framework (JADE), JADE Official Site: http://jade.tilab.com/, last visited: 29.04.2015.

[13] Java Community Process (JCP), JCP Official site: https://www.jcp.org/en/home/index, last visited: 29.04.2015.

[14] L. Jarvenpaa, "Development and Evaluation of HTML5 Agent Framework", Master of Science Thesis, Tampere University of Technology, Tampere, 2013.

[15] E. A. Kendall, P.V. Murali Krishna, C.B. Suresh, C.V. Pathak, "An Application Framework for Intelligent and Mobile Agents", ACM Computing Surveys, Vol. 32, March 2000, pp. 2.

[16] I.J. Timm, M. Berger, S. Poslad, S. Kiran, "International Workshop on Multi-agent Interoperability", Presented in 25th German Conference on Artificial Intelligence, Sept. 2002, Aachen, Germany.

[17] J.J. Tan, Q. Mary, "International Workshop on Multi-agent Interoperability: Open Service Vision of Agentcities", Invited talk in 25th German Conference on Artificial Intelligence, pp. 1-3, Sept. 2002, Aachen, Germany.

[18] S. Willmott, J. Dale, B. Burg, P. Charlton, P. O'Brien, "Agentcities: A Worldwide Open Agent Network", Agentlink News 8, no. LIA-ARTICLE-2001-002, 2001.

[19] S. Turunen, "Productization of an HTML5 Agent Framework", Master of Science Thesis, Tampere University of Technology, Tampere, 2015.

[20] J.P. Voutilainen, A.L. Mattila. K. Systa, T. Mikkonen, "HTML5-based Mobile Agents for Web-of-Things", In WASA2013, Workshop on Applications of Software Agents, 2013.

[21] A. Carzaniga, G.P. Picco, G. Vigna, "Is Code Still Moving Around? Looking Back at a Decade of Code Mobility", In Proceedings of the 29th International Conference on Software Engineering, May 2007, pp. Available at: https://www.cs.ucsb.edu/~vigna/publications/2007_carzaniga_picco_vigna_ICSE.pdf, last visited: 05.05.2015.

[22] FIPA-Compliant Live Systems, FIPA Official Site: http://fipa.org/resources/platforms.html, last visited: 06.05.2015.

[23] FIPA-Compliant Agents, available at: http://www.obitko.com/tutorials/ontologies-semantic-web/fipa-compliant-agents.html, last visited: 06.05.2015.

[24] FIPA Abstract Architecture Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00001/SC00001L.html, last visited: 28.04.2015.

[25] G. Weiss, “Multi-agent Systems: A Modern Approach to Distributed Modern Approach to Artificial Intelligence”, Massachusetts Institute of Technology, pp. 79-95, 1999. Print.

[26] M. Obitko, “Introduction to Ontologies and Semantic Web: Communication between Agents”, available at: http://www.obitko.com/tutorials/ontologies-semantic-web/communication-between-agents.html, 2007, last visited: 08.05.2015.

[27] M. Obitko, “Introduction to Ontologies and Semantic Web: Specification of Conceptualization”, available at: http://www.obitko.com/tutorials/ontologies-semantic-web/specification-of-conceptualization.html, 2007, last visited: 08.05.2015.

[28] M. Obitko, “Introduction to Ontologies and Semantic Web: Ontologies for Agents”, available at: http://www.obitko.com/tutorials/ontologies-semantic-web/ontologies-for-agents.html, 2007, last visited: 08.05.2015.

[29] FIPA ACL Message Structure Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00061/SC00061G.html, last visited: 08.05.2015.

[30] FIPA Agent Management Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00023/SC00023K.html, last visited: 08.05.2015.

[31] FIPA SL Content Language Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00008/SC00008I.html, last visited: 08.05.2015.

[32] FIPA Ontology Service Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00086/XC00086D.html, last visited: 08.05.2015.

[33] FIPA Communicative Act Library Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00037/SC00037J.html, last visited: 08.05.2015.

[34] FIPA KIF Content Language Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00010/XC00010B.html, last visited: 10.05.2015.

[35] FIPA Interaction Protocol Library Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00025/DC00025F.html, last visited: 10.05.2015.

[36] FIPA Agent Message Transport Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00024/OC00024D.html, last visited: 12.05.2015.

[37] X. Haiping, S. M. Shatz, “ADK: An Agent Development Kit Based on a Formal Design Model for Multi-Agent Systems”, The University of Illinois, Department of Computer Science, Chicago, available at: http://www.cis.umassd.edu/~hxu/Papers/UIC/ADK2003.pdf, last visited: 20.05.2015.

[38] Dr. C. Luigi, Dr. D. Jonathan, K. John, “The April Agent Platform: Features and Summary”, available at: http://www.angelfire.com/scifi2/technopapa/2002_02___aap_v08.pdf, Feb. 2002, last visited: 20.05.2015.

[39] S. Poslad, P. Buckle, R. Hadingham, “The FIPA-OS Agent Platform: Open Source for Open Source for Open Standards”, available at: http://fipa-os.sourceforge.net/docs/papers/FIPAOS.pdf, last visited: 20.05.2015.

[40] M. Dejan, B. Markus, B. Ingo, C. John, C. Stefan, F. Barry, K. Kazuya, L. Danny, O. Kouichi, O. Mitsuru, T. Cynthia, V. Sankar and W. Jim, ”MASIF: The OMG Mobile Agent System Interoperability Facility ”, Personal Technologies 2, no. 2 1998, pp. 117-129, available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.3585&rep=rep1&type=pdf, last visited: 21.05.2015.

[41] K. Systa, T. Mikkonen, L. Jarvenpaa, “HTML5 Agents – Mobile Agents for the Web”, In the Proceedings of 9th International Conference on Web Information Systems and Technologies (WEBIST), 2013, Aachen, Germany.

[42] G. Shaw, H. Leon, N. Brenda, Dr.C. Padraig, S. Fergal, and Dr.E. Richards, “Software Agents: A Review”, Technical Report of Trinity College Dublin, Department of Computer Science, 27 May 1997, pp. 2-4, pp. 26-30.

[43] D. Greenwood, “The Foundation for Intelligent Physical Agents”, Whitestein Technologies AG, available at: http://jade.tilab.com/papers/JADETutorialIEEE/JADETutorial_FIPA.pdf, last visited: 21.05.2015.

[44] F. Bellifemine, A. Poggi, and G. Rimassa, “Developing Multi-agent Systems with a FIPA Compliant Agent Framework”, Software: Practice and Experience 2001; 31: 103-128.

[45] Y. Labrou, T Finin, and Y. Peng, “The Current Landscape of Agent Communication Languages”, University of Maryland, Baltimore County, Intelligent Systems and their Applications, IEEE, 1999, Vol. 14, Issue 2.

[46] Node.js, Official Web page for document and download of node.js technology: https://nodejs.org/, last visited: 21.05.2015.

[47] Socket.io, http://socket.io/, last visited: 21.05.2015.

[48] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE-A FIPA Compliant Agent Platform", In Proceedings of PAAM, 1999, Vol. 99, No. 97-108, 1999.

[49] J.L. Austin, "How to Do Things with Words", The William Fames Lectures delivered at Harvard University, 1995, available at: http://www.ling.upenn.edu/~rnoyer/courses/103/Austin.pdf, last visited: 25.05.2015, Print.

[50] FIPA CCL Content Language Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00009/XC00009B.html, last visited: 25.05.2015.

[51] FIPA RDF Content Language Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00011/XC00011B.html, last visited: 25.05.2015.

[52] FIPA Request Interaction Protocol Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00026/SC00026H.html, last visited: 25.05.2015.

[53] FIPA Query Interaction Protocol Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00027/SC00027H.html, last visited: 25.05.2015.

[54] FIPA Request When Interaction Protocol Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00028/SC00028H.html, last visited: 25.05.2015.

[55] FIPA Contract Net Interaction Protocol Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00029/SC00029H.html, last visited: 25.05.2015.

[56] FIPA Iterated Contract Net Interaction Protocol Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00030/SC00030H.html, last visited: 25.05.2015.

[57] FIPA English Auction Interaction Protocol Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00031/XC00031F.html, last visited: 25.05.2015.

[58] FIPA Dutch Auction Interaction Protocol Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00032/XC00032F.html, last visited: 25.05.2015.

[59] FIPA Brokering Interaction Protocol Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00033/SC00033H.html, last visited: 25.05.2015.

[60] FIPA Recruiting Interaction Protocol Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00034/SC00034H.html, last visited: 25.05.2015.

[61] FIPA Subscribe Interaction Protocol Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00035/SC00035H.html, last visited: 25.05.2015.

[62] FIPA Propose Interaction Protocol Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00036/SC00036H.html, last visited: 25.05.2015.

[63] CORBA (Common Object Request Broker Architecture), available at: http://www.corba.org/, last visited: 27.05.2015.

[64] ORB (Object Request Broker) Basics, available at: http://www.omg.org/gettingstarted/orb_basics.htm, last visited: 27.05.2015.

[65] S. Poslad, "Specifying Protocols for Multi-Agent Systems Interaction", ACM Trans. Autonom. Adapt. Syst (TAAS), November 2007, Vol. 2, Issue 4.

[66] FIPA ACL Message Representation in String Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00070/SC00070I.html, last visited: 29.05.2015.

[67] FIPA ACL Message Representation in XML Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00071/SC00071E.html, last visited: 29.05.2015.

[68] FIPA ACL Message Representation in Bit-Efficient Encoding Specification, FIPA Official Site: http://www.fipa.org/specs/fipa00069/SC00069G.html, last visited: 29.05.2015.

[69] D. Mitrovic, M. Ivanovic, Z. Budimac, M. Vidakovic, "Radigost: Inteoperable Web-based Multi-agent Platform", The Journal of Systems and Software, April 2014, Vol. 90, pp. 167-178.

[70] F.L. Bellifemine, G. Caire, and D. Greenwood. "Developing Multi-agent Systems with JADE". Hoboken, NJ: John Wiley & Sons, 2007. Print.

[71] D. B. Lange, M. Oshima, "Seven Good Reasons for Mobile Agents", In Communications of the ACM, Volume 42, Issue 3, March 1999, pp. 88-89.

[72] A. Aneiba, S. J. Rees, "Mobile Agents Technology and Mobility", In Proceedings of the 5th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking, and Broadcasting, 2004.

[73] M. Wooldridge, "An Introduction to Multi-agent Systems", J. Wiley, 2002, pp. 1-8, Print.

[74] J.A. Giampapa, M. Paolucci, K. Sycara, "Agent Interoperation across Multi-agent System Boundaries", In Proceedings of the fourth international conference on Autonomous Agents, 2000, pp. 1-8.

[75] C. Georgousopoulos, O.F. Rana, A. Karageorgos, “Supporting FIPA Interoperability for Legacy Multi-Agent Systems”, In Agent-Oriented Software Engineering IV, pp. 167-184, Springer Berlin Heidelberg, 2004.

[76] Z. AAmir, “Developing a Communication Link between Agents and Cross Platform IDE”, Master Thesis, Hogskolan Dalarna, Sweden, 2010.

[77] Learn JAVA Servlets: Web Application Framework, available at: http://www.tutorialspoint.com/servlets/index.htm, last visited: 01.07.2015.

[78] V. Kelemen, “Accessing a JADE MAS from a Servlet using the JadeGateway class”, 2006, available at: http://jade.tilab.com/doc/tutorials/JadeGateway.pdf, last visited: 01.07.2015.

[79] Apache Tomcat, available at: http://tomcat.apache.org/, last visited: 01.07.2015.

[80] WebSocket, available at: https://www.websocket.org/index.html, last visited: 15.07.2015.

[81] E. German, L. Sheremetov, “Specifying Interaction Space Components in a FIPA-ACL Interaction Framework”, In Languages, Methodologies and Development Tools for Multi-Agent Systems, pp. 191-208, Springer Berlin Heidelberg, 2008.

[82] P. D. O’Brien, R. C. Nicol, “FIPA – Towards a Standard for Software Agents”, BT Technology Journal, Vol 16, No 3, July 1998.

[83] Why standardization is necessary, available at: http://www.zdnet.com/article/why-standardization-is-necessary/, last visited: 29.09.2015.

[84] S. Bhattarai, “Security Framework for HTML5 Agents”, Ongoing Master of Science Thesis, Tampere University of Technology, Tampere, 2015.

[85] H. Suguri, E. Kodama, M. Miyazaki, I. Kaji, “Assuring Interoperability between Heterogeneous Multi-agent Systems with a Gateway Agent”, In Proceedings of 7th IEEE International Symposium on High Assurance Systems Engineering, 2002.