JIAC Systems and JADE Agents Communication

Abdellatif SOKLABI[#1], Mohamed BAHAJ[#2], Ilias CHERTI[#3]
[#] FST Settat/ Department of Mathematics and Computer Science, University Hassan Ist
Settat, 26000, Morocco
[1] abd.soklabi@gmail.com
[2] mohamedbahaj@gmail.com
[3] iliascherti@yahoo.fr

***Abstract*—Many Mobile Agent Systems (MAS) have been developed, but with diverse proprietary Application Programming Interfaces (APIs). This propagation of incompatible APIs implies the complexity of communication and interaction inter-platforms. In a heterogeneous network, an agent may also need to cooperate with agents developed for different platforms. Furthermore, a group of cooperating agents and their interacting pattern are usually dynamic and are unknown at the design stage. As a result, an open and flexible communication and interaction model is needed to ensure the interoperability among mobile agents and mobile agent systems. Agent communication and interaction are achieved through agent communication mechanisms provided by agent systems. Agent communication mechanisms vary considerably from one agent system to another agent system.**

Our work was to find a suitable solution to the communication between both JIAC and JADE agents, which takes into consideration the architecture, API and language with which both SMA were developed.

Keywords-Mobile agents, Agents communication, interoperability, JADE, JIAC

I. INTRODUCTION:

Agents can achieve high-level interoperability by communication at a higher-level of abstraction involving such concepts as beliefs, goals, expectations and intentions. Currently, mobile agent presents a very narrow view of agent communication that does not take full advantage of communication as an interoperability mechanism. We argue that at this critical period of standardization efforts in the broader area of agents, bringing mobile agents and agent communication together is an opportunity not to be missed.

JIAC (Java-based Intelligent Agent Componentware) is a framework that allows the easy development of multi-agent-systems [5], [6], [7], [10], [11]. It presents a specific AOSE [1] methodology and tools that should support and ease the development of agent-based applications. JIAC is built around solid component architecture and uses agent programming language JADL. It is defined as a FIPA compliant agent framework. JIAC provides management and security functionalities as well as a generic scheme for user access. JIAC's agents are based on the BDI (Believe "data", Desire "goal", Intention "adopting plans") concept. That is, to reach a defined goal, an agent may analyze the related data and choose the appropriate plan.

JADE (Java Agent Development Framework) is the most diffuse FIPA compliant agent platforms for mobiles agents applications development [3], [4], [13]. But, it is not interoperable with several of current mobiles agents systems employed for the development of agent-based applications. It is written in Java programming language and is made up of diverse Java packages, which gives to applications programmers many functionalities and interfaces.

ActiveMQ is a message broker which is used to develop a MOM (Middleware Oriented Messages). It supports multiple protocols at several levels for maximum interoperability. It works well with JAVA which is the programming language of both JIAC and JADE. And it exchange messages in asynchronous mode, which is the mode needed to insure communication between JIAC and JADE mobiles agents. So AvctiveMQ was the best MOM to choose in order to provide communication between the two MAS (Mobile Agent System).

There is no notion of an execution environment and the focus is on communication as the means for achieving interoperability [15]. In the latter case, interoperability is akin to effectively exchanging the information and knowledge content of the agents. From this we can conclude that communication is a basic element to ensure the interoperability.

Here, we will try to explain our solution for communication between JIAC agents and JADE agents. Our solution is adapted to the architecture, API and language with which both SMA have been developed. For that we introduce the paper with used component summary and research motivation. Then we describe the overall design of JIAC and JADE mobile-agent systems, Section 2 and Section 3. We present some related work in Section 4. Section 5 presents the Java Message Services. Section 6 discusses how to use ActiveMQ like a

communication layer between JADE and JIAC. Finally, in Section 7, we conclude by presenting a demonstration application of our result.

II. JIAC:

A. What is JIAC?

JIAC is conceived as a comprehensive toolkit for conceiving and deploying multi-agent systems [10] [11] [12] [14]. It presents a specific AOSE [1] methodology and tools that should support and ease the development of agent-based applications. JIAC is built around solid element construction and uses a precise agent programming language and defined as a FIPA compliant agent framework. It provides management and security functionalities as well as a generic scheme for user access.

B. Agents communication in JIAC:

JIAC employs service based interaction. More specifically, while JIAC uses FIPA compliant speech acts for communication, the communication between two JIAC-agents is always handled via service calls. Every service call is wrapped in a so called Meta protocol which automatically takes care of security requirements, error handling, data exchange and negotiation protocols between agents. The actual exchange of messages between agents during one service session however is not restricted, as long as both agents can agree on a common protocol. The default implementation of JIAC's communication components relies on ActiveMQ. The message broker is a component of the AgentNode and supplies messaging between all agents on it and on other nodes.

While service provision always occurs between two agents, the meta protocol also allows to precede the actual provision with a provider selection, where the service user can request a service from many agents and use any suitable protocol such as for example the contract net protocol to select the most suitable provider.

III. JADE:

A. What is JADE?

JADE is an agent platform and development framework [3], [4], [13] compliant with FIPA. Agent platforms are responsible for dealing with agent services such as messaging, development, agent lifecycle management and other common resources. JADE agents execute and share services with other agents present in the container. Our main interest in the JADE platform concerns its messaging capabilities.

JADE provides an agent based platform, index facilitator and ACC which eases the development of agent and its functionalities. An agent is constituted from different synchronized behaviors. JADE employs dedicated methods to dispatch messages among other agents. When agents are in the same platform, the messages are delivered by java local calls according to FIPA specification when crossing platform boundaries. JADE is continuously developed and maintained by Telecom Italia Lab (Tilab) developers, where it was originated first.

B. Communication agents in JADE:

We talk about Inter-platform messaging when two or more agents that reside in different platforms wish to communicate. This communication takes place over JADE’s Internal Message Transport Protocol (IMTP). IMTP techniques currently used by JADE are Java Event passing for intra-container messaging and Remote Method Invocation (RMI) for inter-container messaging. According to the FIPA condition, agents communicate via asynchronous message passing. The Agent class provides methods for agent communication and the ACLMessage class represents exchanged messages between agents. An agent which wants to send a message must create a new ACL Message object that fill its attributes with appropriate values, and then call the agent SEND(). Similarly, an agent ready to receive a message should call RECEIVE() or BLOCKINGRECEIVE() method, both methods implemented by the Agent class. Sending or receiving messages can also be programmed as independent agent actions by adding the behaviors ReceiverBehavior and Sender Behavior to the agent queue of tasks.

IV. SIMILAR WORK:

There was an implementation attempt of a network protocol called Agent Platform Protocol (APP) in JADE in order to realize the interaction between JADE and KODAMA [17]. APP was designed to meet the exact demands of agent interaction over world-wide networks. It works at a lower level than agents do. APP was already implemented in KODAMA. The advantage of the implementation of APP in JADE does not affect the original JADE agent structure. And In order to get KODAMA and JADE agents to work together, they simply assign the Directory Facilitator of a JADE system to the top community in a KODAMA system, so that JADE agents can send SEARCH REQUESTS to the portal agent of the top community if they cannot find them in the JADE system. Also, they use the JADE message form as the standard one where messages are exchanged between JADE and KODAMA. This is because KODAMA agents do not make use of the “address” parameter that is used in JADE agents.

Another work presented Agent Communication Language [18] (ACL) as an interoperability mechanism, made by exploiting that ACLs offer a conceptual framework that can assist in addressing the difficult problem of achieving interoperability between applications. An Agent Communication Language offers some kinds of advantages, like that the declarative nature of most ACLs provides many features that make interoperability easier, such as abstracting away some of the lower-level, more procedural aspects of the systems involved. Moreover, an ACL supports interoperability between static and mobile agents, between mobile agents designed for different agents' platforms and also between mobile agents and static agent information sources. And the higher level of abstraction at which ACLs operate can accommodate multiple paradigms.

An ACL can be considered as a collection of message types each with a reserved meaning. A communication language is not concerned with the physical exchange, over the network, of an expression in some language, but rather with stating an attitude about the content of this exchange. However, this research was conducted before the creation of JIAC which has recently been created and does not take into consideration the architecture and APIs with which the various SMA were created.

V. JMS:

A. What is JMS?

The Java Message Service (JMS) [9], [8] specification provides an Application Programming Interface and a semantics that describe the interface and general behavior of an Enterprise-messaging service. The goal of the JMS is to provide a universal way to interact with multiple heterogeneous Enterprise-messaging services in a consistent approach. All messaging is about the separating of senders and receivers. The messages are sent to a broker and then they are received from a broker in an asynchronous manner, which corresponds perfectly to the asynchronous way of communication between JADE agents.

B. Integrating JMS in JADE :

JMS provides to Java an implementation of MOM (Message Oriented Middleware) or middleware communication messages. However, to use JMS, we must have an external implementation to Java. There are many implementations, both commercial and Open Source. Examples: Websphere MQ IBM, OpenJMS, ActiveMQ which we use here.

There was already an attempt to integrate JMS Message Transport Protocol solution for the JADE Platform to deliver interplatform JADE agent communication Between Platform [8]. In this paper we integrate JMS with JADE and other messaging solution for connection between JADE and JIAC, with respect to the operating principle of JADE, which is possible by adding JMS headers JMS message before JADE sends them. In this way, messages will be easily read and analyzed by the JMS is already integrated in JIAC. JMS can be integrated into JADE like a library that will be called as needed (Fig. 1).

```
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.Destination;
import javax.jms.MessageProducer;
import javax.jms.JMSException;

……..
public class ProducerAgent extends Agent {
……...
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
 msg.setContent( "I am a JADE agent  message" );
 TextMessage message =
session.createTextMessage(msg.getContent());
……..
}
```

Fig.1 Packages and integration of messages in JMS sessions

VI. ACTIVEMQ:

A. Operating principle of ActiveMQ:

ActiveMQ works well with JAVA which is the programming language of JADE; it's a message broker which is used to develop a Middleware Oriented Messages. It supports multiple protocols at several levels for maximum interoperability like AMQP, MQTT, OpenWire, REST, RSS and Atom. In ActiveMQ like any other

MOM messages consists of two parts: the header technique and the data can be in any format. ActiveMQ exchanges messages in asynchronous mode. The sending application of a message and the receiving application of the message don't need to be active simultaneously. The queue receives the message from the sending application and stores it until the receiving application comes and reads the message. It allows the changing of the data messages format to fit the receiving application. We choose AciveMQ to benefit from its reliability. Each message sent by an application is subject to an acknowledgment by the MOM. Every application that consumes a message sends an acknowledgment to the MOM. Coupled with persistence, this mechanism ensures that no message will be lost in the transfer between applications. ActiveMQ [19] has two main operating modes:
• Point to point: an application produces messages and another consumes them. Messages are read by a single consumer. Once a message is read, it is removed from the queue.
• Publish Subscribe (by subscription): messages intensive applications subscribe to a topic (subject, message category). Messages posted to this topic remain in the queue until all applications have subscribed to read the message.

B. Integration of ActiveMQ in JADE:

When using plain old Java to set up the broker, the org.apache.activemq.broker.BrokerService class is one starting point. This class is used to configure the broker and manages its entire life cycle. The BrokerService class is useful when the needing to configure JADE using JAVA for the broker configuration. This method is useful for many situations, in particular when the need of an externally customizable configuration. In many JADE applications, the programmer wants to be able to initialize the broker using the same configuration files used to configure standalone instances of the ActiveMQ broker. For that purpose, ActiveMQ provides the utility org.apache.activemq.broker.BrokerFactory class.

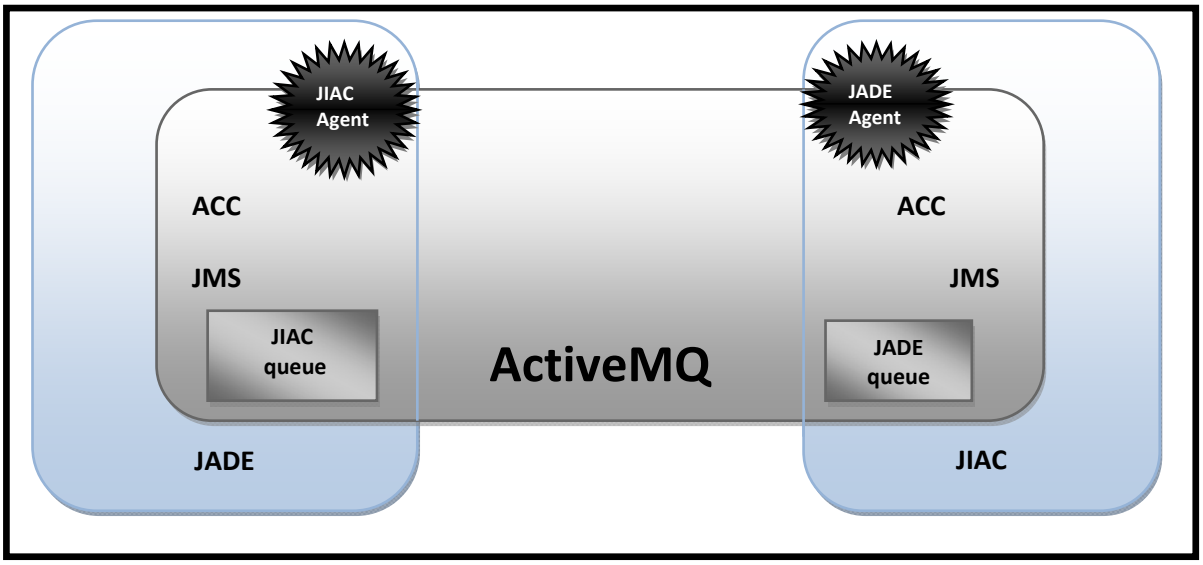
The BrokerFactory class is a utility that makes it easy to create a broker instance simply using an ActiveMQ URI. Depending on the broker URI scheme, the BrokerFactory locates the appropriate factory and uses it to create an instance of the BrokerService class. The most widely used factory is the XBeanBrokerFactory class and is configured by simply passing the XBean-style of URI. ActiveMQ can be integrated in JADE as libraries that will be called as required (Fig.2).

```
import org.apache.activemq.ActiveMQConnection;
import org.apache.activemq.ActiveMQConnectionFactory;
```

Fig.2. imported ActiveMQ libraries

C. Queue structure:

The JMS-ActiveMQ utilizes a queue structure, as illustrated in (Fig.3), which requires each platform to have its own queue for incoming messages. Every SMA may read messages from its queue. To avoid naming conflicts we prefer for all JADE platform queues to be placed under the '/jade' prefix and for all JIAC platform queues to be placed under the '/jiac' prefix. If several platforms of the same type are installed, separation numbers are automatically added to the names of agents (queue_1/jade, queue_2/jade ... queue1 / JIAC, queue_2/jiac ...).



Both JADE and JIAC use the Services for Agent Communication Channel (ACC) stored in ACCService. It contains on the one hand services to forward speech-acts and on the other one service to retrieve temporary stored speech-acts as well as services to set on the storage modus for undeliverable speech-acts. The protocols that implement the provider's role are defined in ACCProvider, service's role is not used.

VII. APPLICATION:

In our application, we sent messages of text type from JADE agents (Producer agent) and receive messages by agents of JIAC (Consumer Agent) using the JMS API and ActiveMQ broker. The programming interface Java Message Service (JMS) allows to send and receive messages asynchronously between applications or Java components. It also allows the messages exchanging between several systems. The JMS model comes from the unification of two modes of communication: • A means of communication from point to point with a temporary storage in a mailbox called Queue, where a producer will leave a message to be read later by a consumer (Fig.4). • A communication mode or event mode publish / subscribe, where zero or more consumers listening on the same broadcast channel called Topic, powered by one or more producer.

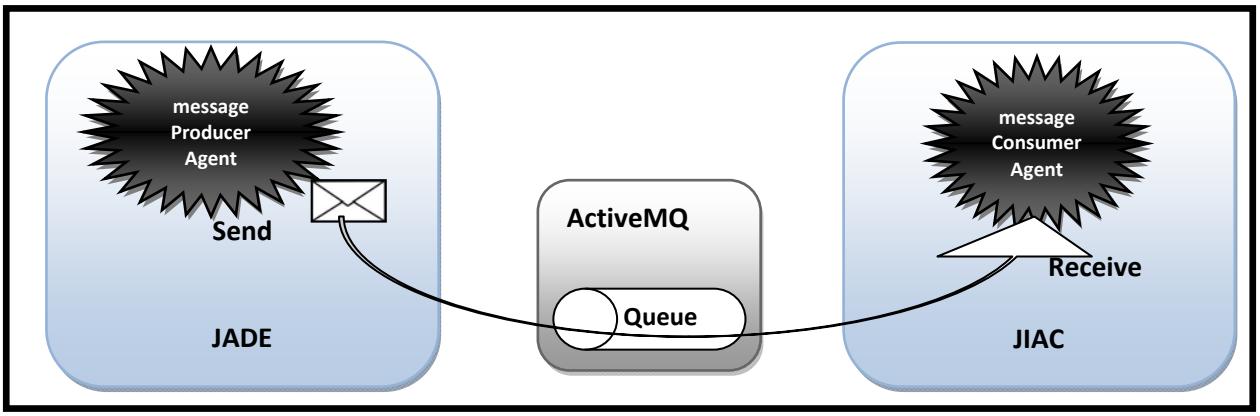


Fig.4 transfer of messages between the Producers agents of JADE and Consumers agents of JIAC

Unlike RPC models (CORBA, RMI and SOAP), the consumers and producers don't share interfaces and exchange information only in the form of messages. A message is defined as the composition of a header, title and content. The latter can be of several kinds: text, binary...

The mechanism for the creation of connection and session is perfectly identical to the creation and reception of messages. It involves two entry points: ConnectionFactory factory of connection and the name of the destination (Queue or Topic). The connectionFactory creates a connection which in turn creates one or more sessions. MessageProducer or a MessageConsumer exists only within a session. It's specific to a destination. A consumer can naturally receive multiple messages, as well as a Producer can produce more messages in the same session.

A. JADE Producer agent:

To send a message, it must have a ConnectionFactory and know other hand, the name of a destination, Queue or Topic. Creating a ConnectionFactory is specific to an implementation, here it is ActiveMQ. The required parameter is the server address and port number (Fig.5).

```
public class ProducerAgent extends Agent {
    private static String broker_url = ActiveMQConnection.DEFAULT_BROKER_URL;
    private static String queue  = "queue1/jiac";
    public static void main( String[] args ) throws JMSException {
    // Getting connection from the server
        ConnectionFactory  CF = new ActiveMQConnectionFactory(broker_url);
        Connection connection = CF.createConnection();
     // Starting the connection
      connection.start();
      // creating a Session.
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
      // the destination queue on the JMS server.
        Destination destination = session.createQueue(queue);
      // sending messages by using MessageProducer
        MessageProducer  producer = session.createProducer(destination);
        ACLMessage acl_msg = new ACLMessage(ACLMessage.INFORM);
        msg.setContent("I am a JADE agent  message");
      // send a text message saying 'I am a producer agent'
        TextMessage  msg = session.createTextMessage(acl_msg.getContent());
        // sending the message
        producer.send(message);
        System.out.println("Sent message '" + msg.getText() + "'");
        connection.close();
    }
}
```

Fig.5 JADE Producer agent

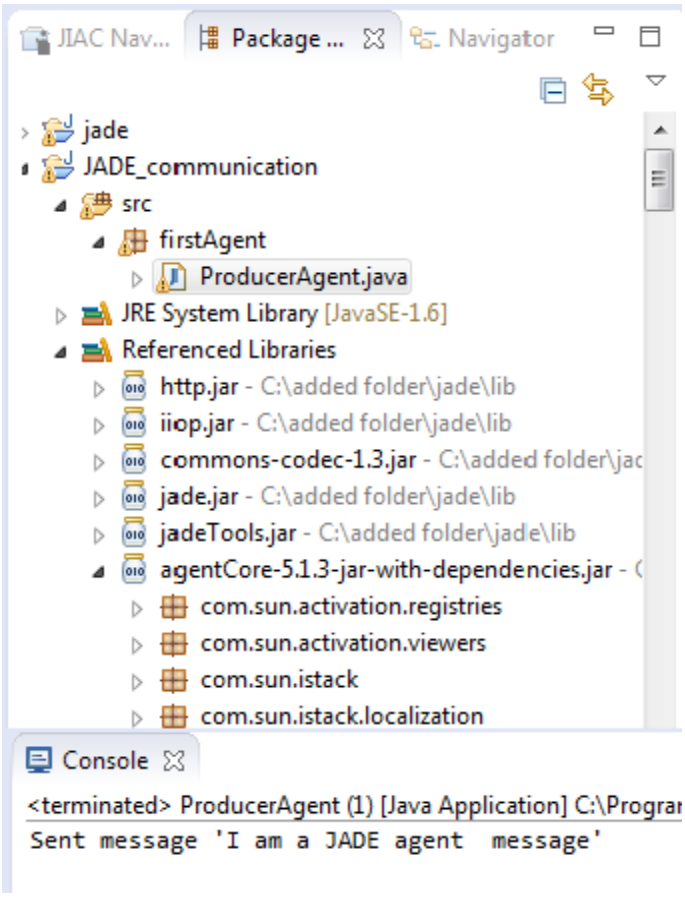


Fig.6 JADE Producer agent execution result

B. JIAC Consumer agent:

The default implementation of JIAC's communication components relies on ActiveMQ [19]. The message broker is a component of the AgentNode (Fig.7) and supplies messaging between all agents on it and on other nodes. The first to do is to add a CommunicationBean (Fig.8) to those agents that want to talk to other agents in the configuration file. Note that, if the user uses the SimpleAgent definition as parent for the configuration, the agent gets a CommunicationBean automatically, so he can omit specifying the property. If he uses a full configuration, he has to provide a communication property. The CommunicationBean registers the IMessageBoxAddress of the agent at the broker, in order to be able to send messages to that agent directly. Furthermore, the CommunicationBean provides some actions that can be used by the agent containing the CommunicatioBean.

```
<?xml version="1.0" encoding="UTF-8"?>

<beans>
 <import resource="classpath:de/dailab/jiactng/agentcore/conf/AgentNode.xml" />
 <import resource="classpath:de/dailab/jiactng/agentcore/conf/Agent.xml" />
 <import resource="classpath:de/dailab/jiactng/agentcore/conf/JMSMessaging.xml"
/>

   <bean name="CommunicationNode" parent="NodeWithJMS">
     <property name="agents">
     <list>
     <ref bean="CommunicationAgent"/>
     </list>
     </property>
   </bean>

   <bean name="CommunicationAgent" parent="SimpleAgent" scope="prototype">
     <property name="agentBeans">
     <list>
     <ref bean="CommunicationBean" />
     </list>
     </property>
   </bean>

     <bean name="Bean" class="communication.CommunicationBean"
scope="prototype">
     <property name="executeInterval" value="1000"/>
   </bean>
</beans>
```

Fig. 7 Consumer AgentNode Configuration

```
public class CommunicationBean extends AbstractAgentBean{

    private static String broker_url = ActiveMQConnection.DEFAULT_BROKER_URL;
    // Queue name from which the message is recaived
    private static String queue = "queue1/jiac";
    public void execute() {
       // Getting connection from the server
        ConnectionFactory CF
            = new ActiveMQConnectionFactory(broker_url);
        Connection connection = CF.createConnection();
        connection.start();
        // Creating session
        Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
        // Getting the queue 'queue1/jiac'
        Destination dest = session.createQueue(queue);
        // MessageConsumer is used for receiving messages
        MessageConsumer consumer = session.createConsumer(dest);
        //Receive the message.
        Message msg = consumer.receive();
       // Printing the message
            TextMessage tMessage = (TextMessage) msg;
            System.out.println("Received message '" + tMessage.getText() + "'");
        connection.close();
    }
}
```

Fig.8 JIAC agent CommunicationBean

The Figure 9 shows a screenshot of the JIAC project structure with different used libraries and the result of the execution of the agent consumer which gets the same message that was sent by the JIAC agent producer.

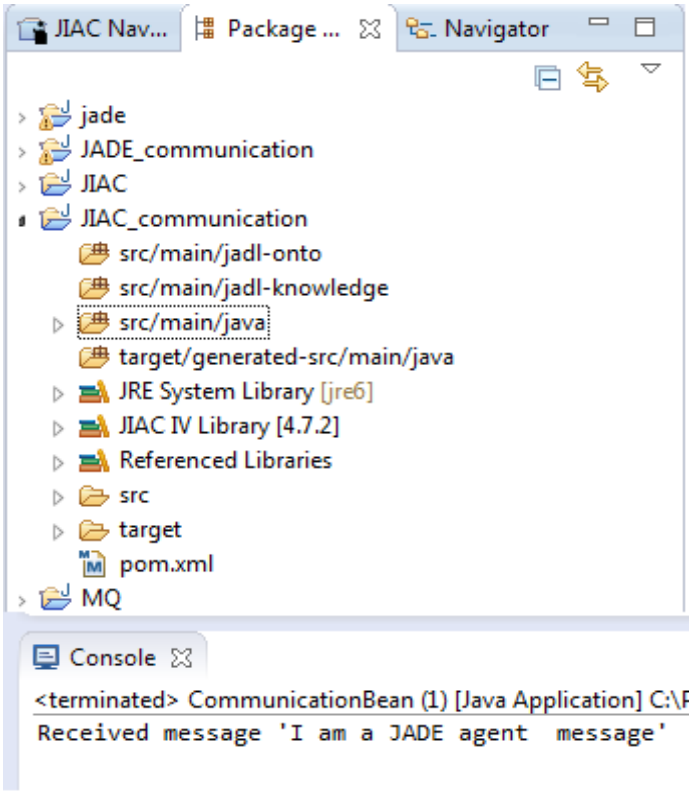


Fig.9 JADE Producer agent execution result

VIII. CONCLUSIONS AND FUTURE WORK:

The software of agents offers a new model for building very large scale distributed heterogeneous applications that focus on the interactions of autonomous, cooperating processes that can adapt to humans and other agents. As such, communication is the solution to realize the potential of this new model, just as the development of human language was critical to the development of human society and culture. This holds for mobile agents as well as for static ones. Agents use an Agent Communication Language to communicate information and knowledge. In this paper, we have to communicate the agents of the most used MAS in the world and the agents of the newest framework of MAS creation, that always in ongoing amelioration by a very active searcher community. The work is not going to stop at this stage, as we continue through the Securing agent communication before proceeding to push research in the way of interoperability which was our original goal.

REFERENCES:

[1] M. Gaglio, S. Garro and A. Seidita, "Method fragments for agent design methodologies, from standardization to research" In Agent-Oriented Software Engineering, April 2007, pp. 91-121.

[2] A. Grimstrup, R. Gray and D. Kotz, "Toward Interoperability of Mobile-Agent Systems" in sixth IEEE International Conference on Mobile Agent , Barcelona, Spain, October 2002, pp. 106-120

[3] R. Gupta and G. Kansal, "A Survey on Comparative Study of Mobile Agent Platforms" in International Journal of Engineering Science and Technology (IJEST). ISSN: 0975-5462 Vol. 3 No. 3 March 2011.

[4] F. Bellifemine, A. Poggi and G. Rimassa, "JADE: A FIPA compliant agent framework".

[5] B. Hirsch, T. Konnerth and A. Hebler. "Merging agents and services in the JIAC agent platform" in Languages, Tools and Applications, pp. 158-184. Springer, 2009.

[6] T. Kuster "The Visual Service Design Tool" Version 1.4.2, march 2012.

[7] P. S. Tan, "Automated generation of JIAC agentbeans from BPMN diagrams". Diploma thesis, Technische University at Berlin, November 2011.

[8] E. Curry, D. Chambers and G. Lyons, "A JMS Message Transport Protocol for the JADE Platform"

[9] Sun Microsystems "Java Message Service: Specification," 2001.

[10] T. Erdene-Ochir and M. Patzla "Programming Multi-Agent Systems, chapter Collecting Gold: MicroJIAC Agents in Multi-Agent Programming Contest", pp. 251-255. Springer Berlin/Heidelberg, 2008.

[11] R. Sesseler, "Eine modulare Architektur futur dienstbasierte Interaktionen zwischen Agent", PhD thesis, Technische University at Berlin, January 2002.

[12] T. Kuster, M. Lutzenberger, A. Hebler, and B. Hirsch, "Integrating Process Modelling into Multi-Agent System Engineering" in Multi agent and Grid Systems, International Journal, pp. 105-124, January 2012

[13] F. Bellifemine, G. Caire, T. Trucco and G. Rimassa, "JADE PROGRAMMER'S GUIDE", last update: April 2010.

[14] B. Hirsc, S. Fricke, O. Kroll-Peters, T. Konnerth, "Agent Programming in Practice —Experiences with the JIAC IV Agent Framework"

[15] Y. Labrou, T. Finin and Y. Peng, "The Interoperability Problem: Bringing together Mobile Agents and Agent Communication Languages"

[16] K. Takahashi, G. Zhong, S. Amamiya, T. Mine and M. Amamiya, "Communication Protocol for Agent Platform: Agent Platform Protocol", In Proc. of Tenth Workshop on Multiagent and Cooperative Computation, pp.30-37, November 2001.

[17] K. TAKAHASHI, G. ZHONG, D. MATSUNO, "Interoperability between KODAMA and JADE using Agent Platform Protocol"

[18] Y. Labrou, T. Finin, and Y. Peng. "The interoperability problem: Bringing together mobile agents and agent communication languages" In Proceedings of the Hawaii International Conference on System Sciences, Maui, Hawaii, January 1999.