Journal of Theoretical and Applied Information Technology
10th September 2013. Vol. 55 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

JIAC SYSTEMS AND JADE INTEROPERABILITY

[1]ABDELLATIF SOKLABI, [2]MOHAMED BAHAJ, [3]JAMAL BAKKAS

[1] PhD student, Department of Mathematics and Computer Science, University Hassan I, Settat, Morocco
[2] PhD, Departments of Mathematics and Computer Science, University Hassan I, Settat, Morocco
[3] PhD student, Department of Mathematics and Computer Science, University Hassan I, Settat, Morocco
E-mail: [1] abd.soklabi@gmail.com, [2] mohamedbahaj@gmail.com, [3] jbakkas@yahoo.fr

ABSTRACT

In the literature various Mobile Agents Systems (MASs) have been created, but with different proprietary technologies and application programming interfaces. In a heterogeneous network, an agent may need to migrate from one mobile agents system to another. Furthermore, not all the MASs are able to receive foreign agents, because a few of the MASs attempt to realize the FIPA Ontology Agent, and each one adopts a particular point of view of the interoperability problem. As a result, a system of transferring and transformation is needed to permit an agent created in one multi-agent system to continue realizing its tasks on another multi-agent system.

Our work was to find an algorithm to transform and transfer mobile agents between JIAC and JADE taking the information contained in a JIAC agent to create its equivalent in JADE and saw to that, by migrating an agent from one of the two systems with its functions, attributes, and its added features, it can operate on another multi-agents system, based on our previous solution to the communication between both JIAC and JADE agents, which takes into consideration the application programming interfaces and the architecture language with which both systems were developed.

Keywords— *Mobile agents, mobile agents system, interoperability, JADE, JIAC*

1. INTRODUCTION:

The technology of mobile agents is an effective solution for many industrial problems. This generated the development of many mobile agents systems, with different proprietary Application Programming Interfaces (API), so the majority of mobile agents systems are not interoperable with currently available MASs. This proliferation of incompatible APIs implies that agents developed for one agent platform cannot be run on, let alone migrate to, a system with a different agent platform. In our opinion, interoperability of platforms is essential for mobile agents to become ubiquitous technology. This paper represents the results of research conducted by trying to answer questions like; is that the agents of MAS continue their execution in other MAS, or what are the needs of an agent JIAC for it to continue running in JADE?

In the literature, MASIF Mobile Agent System Interoperability Facility [8] and Grid Mobile Agent System (GMAS) [7] were the most interesting interoperability structure that allow foreign agents to execute in a foreign Mobile Agents System by translating the foreign agent's API into the local platform's API, However none of them have the ability to encourage many systems to adopt them, because they don't give a pragmatic result, and their approaches are not adapted to the most multi-agents system APIs, and the technologies with which they have been developed.

The authors of the article "Toward Interoperability of Mobile-Agent Systems" [7] have demonstrated that interoperability between different mobile agent systems is possible, and they insisted on the possibility of transforming agents between different multi-agents systems in their conclusion. The goal of this paper is to advance the research on the interoperability of multi-agents systems, by using the most diffuse FIPA compliant agent platforms JADE [for the development of industrial applications] and JIAC that use the newest and advanced technologies. However, both of them are not interoperable with each other.

Here, we will explain how to adapt a JIAC agent to function in a JADE system. For that we will begin the paper with the interoperability research motivation. Then in section 3 we will describe the state of art the mobile agents' interoperability. In section 4 we describe mobile agents' structure

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

while specifying JIAC and JADE agents' particularities. In Section 5 we will compare JADE and JIAC agents. Section 6 talks about our previous work in JADE and JIAC communication. Section 7 emphasizes the importance of taking the research results in the migration of a mobile agent in the agent's migration between different mobile agent systems. Section 8 details how to transfer and transform a JIAC agent to a JADE agent. Finally, in Section 9, we conclude by presenting a demonstration application of our result.

2. THE INTEROPERABILITY

There is many definition of interoperability [2] [3]. While the interoperability was initially defined for IT systems or services and only allows for information to be exchanged, a more generic definition of this could be that the IEEE Glossary that defines interoperability as the ability of two or more systems or components to exchange information and to use the information that has been exchanged. Again, the research distinguishes between two types of interoperability.

Syntactic interoperability [1] [2] [3] is the lack of ability of a systems to communicate and exchange data with other systems. Specified data formats, communication protocols and the like are fundamental. This is also true for lower-level data formats, such as ensuring alphabetical characters are stored in the same variation of the Americans Standard Code for information interchange, or a Unicode format in all the communicating systems. Syntactical interoperability is a necessary condition for further interoperability.

Semantic interoperability [4] [5] [6] is the lack of the ability of two or more computer systems to exchange information. In other words, it is the lack ability to automatically interpret the information exchanged meaningfully and accurately in order to produce useful results as defined by the end users of both systems. To achieve semantic interoperability, both sides must refer to a common information exchange reference model. The content of the information exchange requests are unambiguously defined; what is sent is the same as what is understood.

Whereas, the interoperability of a mobile agents system is it property, to work with others mobile agents systems, present or future, without any restriction access or implementation, and the capability to permit mobile agents to exchange data via a common set of exchange formats, to read and write the same file formats, and to use the same protocols.

3. STATE OF THE ART PROCESSING AGENTS BETWEEN DIFFERENT MASS

MASIF [8] was presented in the literature, as a standard for mobile agent systems, that was an early attempt to standardize interoperability between industrial mobile agents systems, that even though popular in the past, still has not caught on it defined a standard API and required all platforms that wish to interoperate, to then implement the common API. MASIF provides the features used in the transport of agent standardized information which is the first level of interoperability. How the system deals with the parameters internally, once the information is transferred from one mobile agent system to another is an implementation subject and not addressed by MASIF. Such information includes agent profile which describes the serialization, language, and other agent requirements on the host system. MASIF permits the MAS to understand the requirements that the agent has on its native system because they consider that it is the first step in end to end interoperability. These approaches, however, have failed to encourage many systems to adopt the API. Since MASIF is about interoperability between agents systems written in the same language, it does not standardize local agent operations such as serialization/deserialization, agent interpretation, and execution.

Another interesting interoperability structure is the Grid Mobile Agent System (GMAS) [7] that allows foreign agents to execute in a non-native mobile agents system by translating the foreign agent's API into the local platform's API. Instead of creating pair-wise translations of the multi-agents system APIs, they define a single common interoperability API (IAPI) that supports agent registration, lookup, messaging, launching, and mobility. Then each group provided translators between their multi-agents system API and the IAPI. In this manner, it was not necessary to rewrite each MAS to conform to a new API, but to write translators from each MAS to the IAPI, and conversely, from the IAPI to the multi-agents system to support these specific agent operations. Since each mobile agents system has its own communication and directory services, they rely upon a common substrate to provide communication and discovery services in a heterogeneous environment instead of construct mappings between them.

The GMAS API provides methods to create an agent either by launching a new agent or by cloning the current agent on a remote host. When launching

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

a new agent, the agent's initial state must be provided to the system as a parameter. Cloning an agent requires that the state of the agent be moved to the destination.

One of the main advantages of using Java for mobile-agent programming is the ability to use object serialization for packaging an agent before shipment to another host. Any agent that implements Java's Serializable interface can be cloned through the GMAS API.

4. MOBILE AGENTS STRUCTURE

Each agent has a execution flow to be able to take the initiative to perform tasks. To talk about the migration of an agent, it is important to know what to transfer with the agent. A mobile agent instance is an entity that has five attributes: its status, its implementation, its user interface, its identifier and its authority [12] [13]. When an agent moves through the network, it carries its attributes Table 1.

Table 1: the agent attributes with their descriptions

The attribute	Description
The state	The state of an agent can be seen as a snapshot of its execution that allows it to resume execution when it reaches its destination.
The implementation	A code that represents the sequence of instructions defining the static *behavior* of the mobile agent, it allows it to run when it moves through the network.
The interface	An interface that allows other agents and other systems to interact with the agent.
The identifier	Each agent has a unique identifier during its life cycle, allowing it to be identified and located. Since the identifier is unique, it can be used as a key in the operations that requires a means to reference a particular agent instance.
The authority	An authority is an entity whose identity can be authenticated by any system to which it is trying to access. The identity consists of a name and other attributes.

4.1 JADE agents properties

4.1.1 Agent class:

A JADE agent is simply an instance of a defined Java class that extends the base Agent Class. This implies the inheritance of features to accomplish basic interactions with the agent platform like registration, configuration or remote management and basic set of methods that can be called to implement the custom *behavior* of the agent like send or receive messages, use standard interaction protocols, register with several domains. The computational model of an agent is multitasking, where *behaviors* are executed concurrently. Each service provided by an agent should be implemented as one or more *behaviors*. A scheduler, internal to the base Agent class and hidden to the programmer, automatically manages the scheduling of *behaviors* [14].

4.1.2 Agent state:

A JADE agent can be in one of several states, according to Agent Platform Life Cycle in the FIPA specification. These are detailed in the Table 2.

Table 2: the agent states with their descriptions

The agent State	Description
INITIATED	The Agent object is built, but hasn't registered itself yet with the MAS, has neither a name nor an address and cannot communicate with other agents.
ACTIVE	The Agent object is registered with the MAS has a regular name and address and can access all the various JADE features.
SUSPENDED	The Agent object is currently stopped. Its internal thread is suspended and no agent behavior is being executed.
WAITING	The Agent object is blocked, waiting for something. Its internal thread is sleeping on a Java monitor and will wake up when some condition is met.
DELETED	The agent is definitely dead. The internal thread has terminated its execution and the Agent is no more registered with the MAS.
TRANSIT	A mobile agent enters this state while it is migrating to the new location. The system continues to buffer messages that will then be sent to its new location.

4.1.3 Agent execution:

The JADE framework controls the birth of a new agent according to the following steps: the agent constructor is executed, the agent is given an identifier from the jade.core.AID class, it is registered with the AMS, it is put in the ACTIVE state, and finally the setup() method is executed. The setup() method is therefore the point where any application-defined agent activity starts. The programmer has to implement the *setup*() method in order to initialize the agent. When the setup() method is executed, the agent has already been

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

registered with the AMS and its Agent Platform state is ACTIVE. This initialization procedure is used to modify the data registered with the MAS or set the description of the agent and its provided services and if necessary, register the agent with one, more domains or add tasks to the queue of ready tasks using the method addBehavior(). These *behaviors* are scheduled as soon as the *setup*() method ends. The *setup*() method should add at least one *behavior* to the agent. At the end of the *setup*() method, JADE automatically executes the first *behavior* in the queue of ready tasks and then switches to the other *behaviors* in the queue by using a round-robin non-preemptive scheduler. The addBehavior(Behavior) and removeBehavior(*Behavior*) methods of the Agent class can be used to manage the task queue.

The Agent.takeDown() method is executed when the agent is about to go to DELETED state, i.e. it is going to be destroyed. The takeDown() method can be overridden by the programmers in order to implement any necessary cleanup. When this method is executed the agent is still registered with the MAS and can therefore send messages to other agents, but just after the takeDown() method is completed, the agent will be de-registered and its thread destroyed. The intended purpose of this method is to perform application specific cleanup operations, such as de-registering with Directory Facilitator agents [14].

4.2 JIAC agents properties

4.2.1 Agent components

An agent consists of many types of components and is described using a properties file. This properties file consists of a set of entry keys that are separated by an equal sign. If more than one value is assigned to a key, they must be separated by using separators such as blank, tabulator, comma or semi-colon. Each entry in the property file must begin at a newline.

The properties files are used for configuration of the agent and can be globally accessed by each component through the properties. Additionally, to properties that are common to all components, there are components' specific properties files, which are used by a component for internal configuration purpose. Properties for the agent core are defined for multiples functions like permissions, to control the access to the agent, the agent's name that is automatically given by the platform's manager and the initial state of the agent.

The agents consist of a set of ontologies, rules, plan elements, and initial goal states. The state of the world is represented within a so-called fact base which contains instantiations of categories which are defined in ontologies. AgentBeans contain methods which can be called directly from within Jadl, allowing the agent to interact with the real world, via user interfaces, database access, robot control, and others components [16].

4.2.2 Goals

In JIAC, a StateGoal is a single goal of the agent, which describes the state of the world to be reached. And a ServiceGoal is a goal to execute protocol for a service. The execution of goal can be prioritized using the method setPriority. The priority is an integer value comprised between 1 and 10, normally set to 4. Goal with higher priority values are chosen first for execution. Components can determine and set up new goals to control the *behavior* of agents, by passing messages to the control components. To this end, a goal should first be specified and sent to the GoalSelectionRole using the ChangeGoalMessage. The method of the DefaultApplicationBean addGoa()l is used to setup a new goal, which in addition to the goal itself has as parameters a reference to the goal and the context.

4.2.3 Agent State

The agent as well as its components has a discrete state during their whole life-cycle. The state of the agent determines its *behavior*. The following table gives an overview of all the component states, with their explanation Table 3.

Table 3: JIAC agent's component states with their explanations

JIAC agent component states	Explanation
STATE VOID	Start and end state. The component is not part of the agent
STATE STOPPED	Possible component's state after the first transition from
STATE INITIATED	All prearrangement before transition in an active state is doing here
STATE ACTIVE	Component is part of the agent and active
STATE SUSPENDED	Temporary suspended execution of component
STATE STEPPING	Execution step by step
STATE SERIALIZED	The component is prepared for serialization.
STATE TRANSIENT	The agent is in migration between two platforms

The method changeState() is used for changing agent's state during its lifetime. The agent's state is

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

set to the unknown value if the method changeState fails.

4.2.4 FactBase

Unlike components where communication happens through messages, the agent's factbase is accessed directly using methods. The advantage is in the efficiency of communication and hence the consistency of the knowledge base. Directly accessing the factbase avoid the long delay that result from waiting for request responses. The manipulation of the fact base is done through the variable factAccess, which is an instance of the class FactAccess in de.dailab.control.component.

The variable factAccess is declared in the class ControlComponent and should be used by all components in the agent's architecture to manipulate the factbase. There is exactly one factbase per agent.

4.2.5 Agent execution

The capabilities of an agent for deliberative actions are described by plans consist of an execution part (a set of conditions that must be satisfied before or during the execution) and the resulting effect after a successful execution. Adopting a plan for execution in order to achieve one of its goals is known as an intention.

The interior of a JIAC agent has certain useful characteristics as well. Each agent has to provide basic functionalities in order to deal with its beliefs, plans, goals and intentions. Each of these capabilities is implemented by reusable components that can be configured into an agent that works even at runtime. Thus, the definition of a specific agent type is done defining a configuration file that contains all components belonging to the agent [16].

5. JADE AGENT AND JIAC AGENT COMPARISON

If we ignore the differences in the name of some states, the JADE agents and JIAC agents pass through the same states during their life cycles. Also, both JADE and JIAC respect FIPA specifications to identify the agents. But the manner of agent identification differs in the two systems. On the one hand, a JADE agent is identified by a globally unique name that concatenates the local name plus the '@' symbol, the home agent platform identifier, and a set of agent addresses and a set of resolvers (the white page services with which the agent is registered). In other hand, JIAC does not use the concept of agent identifier directly, but it contains some methods in *KAgentName* JAVA class that can manage the agent identifier. For examples it uses *getAgentIdentifier*() method that converts a JIAC agent object to an agent identifier and *parseAgentIdentifier*() that converts an agent identifier, to an object of JIAC category Agent. Another difference between the two systems is that a JADE agent execution is due to the sequential or parallel execution of its *Behaviors*, while executing JIAC agent is structured according to the operator plans that consist of an execution part, a set of conditions that must be satisfied before or during the execution and the resulting effect after a successful execution.

So the complete JIAC (or JADE) agent transformation as an agent of JADE (or JIAC) is possible, but requires a lot of effort. The adaptation of the agent so that it fits into another non-native multi-agent system to communicate and collaborate with its agents is less complicated and more convenient.

6. JIAC SYSTEMS AND JADE AGENTS COMMUNICATION

In previous work we found a suitable solution to the communication between both JIAC and JADE agents, which takes into consideration the architecture, APIs and language with which both MAS were developed, and by integrating ActiveMQ and The Java Message Service (JMS) [9], [8] in JADE. 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 mobile agents. So AvctiveMQ was the best MOM to choose in order to provide communication between the two MAS (Mobile Agent System).

The advantage of JMS is that it provides an API and a semantic that describes the interface and general *behaviors* 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.

The search result was proven by an application in which we have exchanged messages between JADE agents and JIAC agents as shown in Fig 1. We used the result of communication between the two MAS to transfer agents between JIAC and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

JADE, and also to assure communication between agents of different MAS (Figure1).

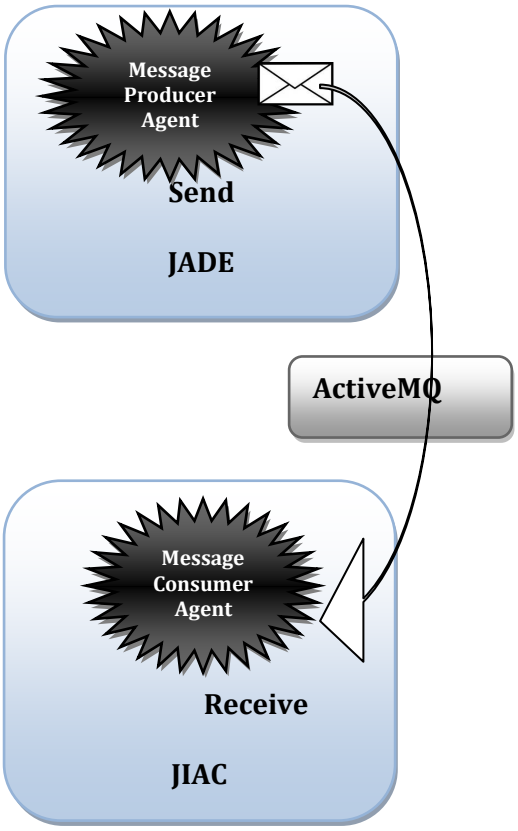


Figure 1: *transfer of messages between the Producers agents of JADE and Consumers agents of JIAC*

7. THE AGENT'S MIGRATION BETWEEN DIFFERENT MASS:

While seeking answers to the question that we asked about the data the agent will need to work in the other MAS. We found answers in the search results on the migration of mobile agents. We conclude that the transfer of mobile agents between different MAS is actually a generalization of agent's migration between parts of the same MAS. In general the state of the art of mobile agent's migration research has distinguished between two types of migration:

Strong migration, where the complete of the agent (i.e. code, data and execution unit) migrates to the new site. For the actual migration, the agent is suspended or captured before being transferred. Once at the remote site, it restarts execution at the previous checkpoint, maintaining the state of the process. Another option proposed is to stop the execution of the agent before the migration and to create an identical remote copy at the remote site.

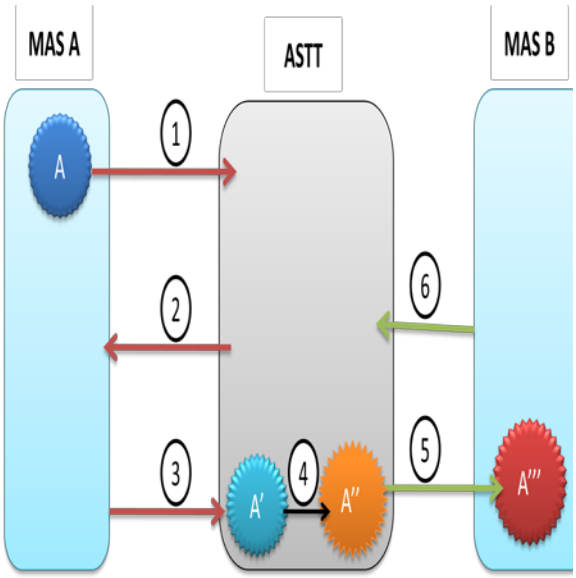
Weak migration, only transfer the agent's code and data. On the destination site, the agent restarts its execution from the beginning by calling the method which represents the input of the agent execution, and test the agent execution context.

So we can extend this distinction between different modes of mobile agents' migration and project it on the transfer of mobile agents between different mobile agents systems, to say that there are two transfer modes. Strong transfer where the complete agent is transferred to the new MAS, weak transfer is where only the code and data of the agent are transfered. The algorithms developed and presented in this paper allow capture and transfer of the executed state of agents that we can consider as a strong migration, just as a migration between different multi-agents systems.

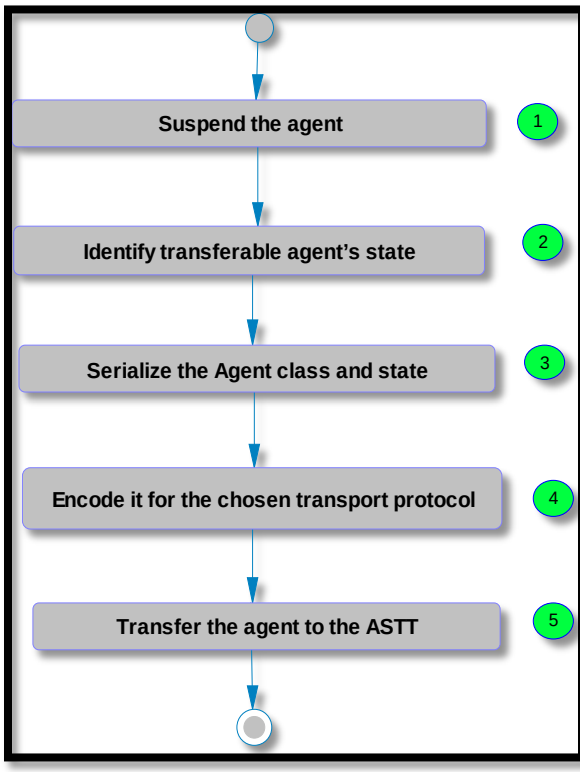
8. THE TRANSFER AND PROCESSING OF AGENTS

Once an agent needs to migrate to new MAS, it sends a message to the Agents System of (ASTT) Transformation and Transfer and transfer agents Fig 2a (1). Then the ASTT sends a message that contains the agent identifier to the agent's current MAS (2). When the source MAS receives the transfer request it executes the algorithm described in Figure 2b: starting by suspending of the agent in 2b (1), second by identifying transferable agent state 2b (2) to serialize the agent class and its condition 2b (3), and encoding it according to the transport protocol 2b (4), before sending it in a serialized state to the ASTT that transform and transfer the agent to the destination MAS. Once the destination system receives the ASTT 2b (5), the ASTT records the identifier and the location, and adds the header of the MAS destination to the agent serialized code 2a (4) before sending it to the destination MAS. Then run the algorithm presented in Figure 2c begins by decoding the agent 2c (1), deserializes its state and class 2c (2), to instantiated it in 2c (3) and restores its state in 2c (4), and finish boost its executing in 2c (5). The MAS destination must resend the identifier of the agent copy on the new system to the ASTT, to allow it to find the new agent in case it needs to recover it.

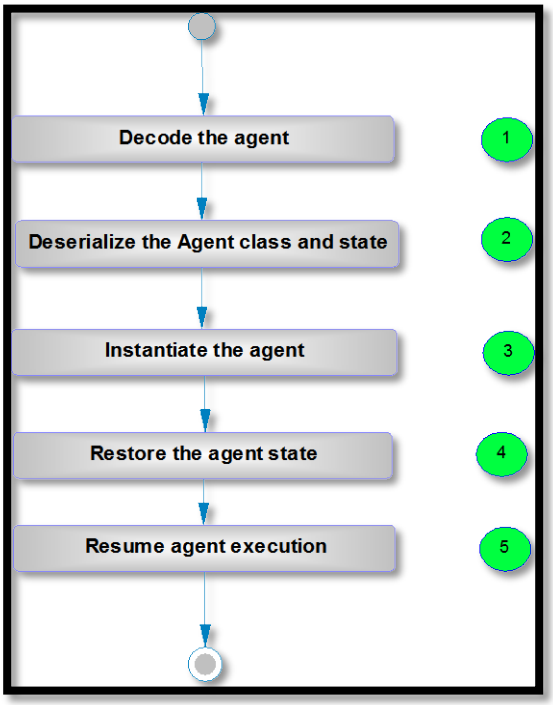
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



a) Agent transfers and transformation (sender side).



b) Initiating agent transfer (sender side).



c) Receiving an agent (receiver side).

```
de.dailab.jiac.agent.beans=\
de.dailab.control.component.FactBean \
de.dailab.control.component.GoalBean \
de.dailab.control.component.SelectionBe
an \
de.dailab.control.component.ExecutionBe
an \
de.dailab.control.component.Communicati
onBean \
de.dailab.control.transport.component.J
VMCommunicationBean \
de.dailab.control.transport.component.A
ctiveMQCommunicationBean \
de.dailab.control.component.TimerBean \
de.dailab.control.debugging.component.D
ebuggerBean \
de.dailab.jiac.component.CommunityBean
\
de.dailab.jiac.component.AMSUserBean \
de.dailab.ASTT_contactor.component.Extr
aBean
```

Figure 2: Algorithms for agent transfer

Each JADE agent belongs to a Container, so a receiving container must be created on JADE by which the ASTT transfers transforming agents from other mobile agents systems. The class of the source agent system is needed to instantiate the transferred agent. since it is possible that classes are

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

needed after instantiation of the agent, a solution is to provide to the transferred agent all classes need in the new MAS, by calling the code source of JIAC, as an *EXTERNAL JAR* in the JADE project. In this way the JIAC agents will be executed using JIAC classes, they will be identified on JADE.

Both JADE and JIAC contain serialization methods that allow serializing and deserializing the agent class and state. When the system attempts to move the serialized agent to a new host, the native system retrieves from the agent a container holding objects that represent the agent's state. Each object's value in the container is then converted to XML notation containing its name, data type and value. Then to restore the agent state, this XML notation is parsed, and the state of the agent is reconstructed using JADE *deserialization* methods.

9. APPLICATION:

We will show the results of our research with an application that create an agent in JIAC and transform it using ASTT in order to move the agent on JADE. This application is based on our previous result which we detail in our article "communication between the JADE agents and JIAC" [11], we achieve it using the followed the steps.

9.1 prepare the agent to be transferred from JIAC

We will begin begun by executing the algorithm of agent transfer presented in the Figure 2b. We have chose JIAC as the agents originate system. We used JIAC *Aclass* class that implements the class *java.io.Serializable* a special class to represent *serializable* JAVA objects stored in attributes and used for the communication between JADE and JIAC in our precedent work [11], instead of the transfer protocol that have used on the migration service. It can be loaded in the plan-library by including the file *de/dailab/jiac/knowledge/MigrationService.know* in the properties file.

To simplify the management of the agents transferred from JIAC to ASTT, we have added a controlling agent that will execute the agent's transfer algorithm in JIAC and communicate with the ASTT. The properties file for the controlling agent is shown in the Figure 3.

Figure 3 the properties file for the controlling agent

The agent is created in an active state. It has the permission to monitor speech acts and is mobile. While migrating, the agent should transport the archive *ExtraBean.jar* that contains all supplementary information about the agent as shown in Figure 4.

```
(ASTT_contactor.agent):
de.dailab.jiac.agent.permissions=mo
bile
de.dailab.jiac.agent.permission.mon
itor
de.dailab.jiac.agent.state=active
de.dailab.jiac.agent.codebase=backs
lash
de/dailab/ASTT_contactor/ExtraBean.
jar
```

Figure 4 The Archive Extrabean.Jar Content

We have already proven that the agent conversion between different MASs is based on the agents migration principles. We have exploited this result to simplify how agents are sent to the ASTT. For this purpose we have added following files to the plans and services libraries of JIAC Table 4.

Table 4: the plan added files and services libraries

The added component	Description
MigrationService	The service for agent migration
TransmitAgentProvider	Providing script that implements the service defined in *TransmitAgentService*
TransmitAgentService	The service responsible for agent's transmission from the source to the destination platform.
MigrationSupport	Plan-element that is used by the *MigrationService*

For all management functionalities provided, the manager agent requires protocol plan-elements for the provider as well as for the user role. Services including those functionalities for migration are also required by the MAS as it is shown in Figure 5.
Figure 5: Services migration

9.2 ASTT

To simplify the transformation of the agents, we have integrated Camel with ActiveMQ as shown in Fig 6. Camel supports the Message Translator in the routing logic, by using a beans to perform the transformation, or by using *transform*() in the DSL. Here we will give the principles interfaces, and their prototypes of essentials methods, that allow

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the management of the transfer and the transformation of agents on the ASTT. The ASTT's places registration, agents' registration, searches, transformations, statute verifications, receptions and agents' locations lists of declared like it was shown in the Fig 7.

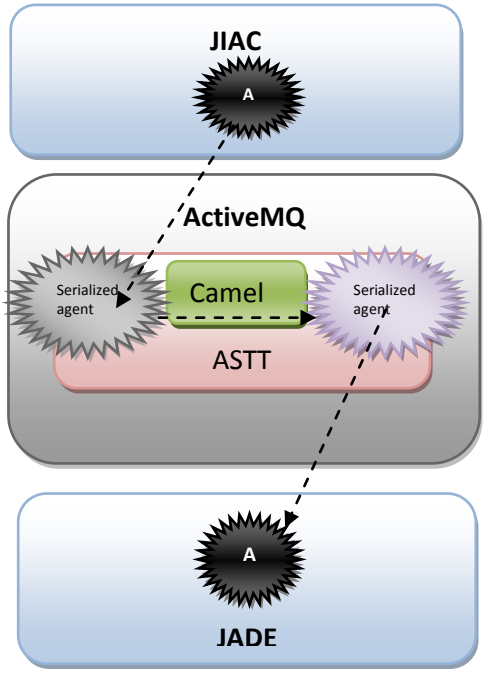


Figure 6: Agents transfer and transformation using ActiveMQ, Camel and ASTT

```
de.dailab.control.role.ServiceLibraryRo
le.services=\
de/dailab/jiac/knowledge/APService.know
\
de/dailab/jiac/knowledge/AMSService.kno
w \
de/dailab/jiac/knowledge/DFService.know
\
de/dailab/jiac/knowledge/MigrationServi
ce.know \
de/dailab/jiac/knowledge/TransmitAgentS
ervice.know
```

```
interface ASTTAgentFinder {

void register_agent (Name agent_name, Location
agent_location, AgentProfile agent_profile);
void register_place (string place_name,
Location place_location);
Locations lookup_agent (Name agent_name,
AgentProfile agent_profile);
Location lookup_place (string place_name);

}

interface ASTTAgentSystem {

Name transfom_agent (Name agent_name,
AgentProfile agent_profile, OctetString agent,
string place_name, Arguments arguments,
ClassNameList class_names, ASTTAgentSystem
class_provider);
OctetStrings fetch_class(ClassNameList
class_name_list, AgentProfile agent_profile);
Location find_nearby_agent_system_of_profile
(AgentProfile profile);
AgentStatus get_agent_status(Name agent_name);
AuthInfo get_authinfo(Name agent_name);
ASTTFinder get_ASTTFinder();
NameList list_all_agents();
Locations list_all_places();
void receive_agent(Name agent_name,
AgentProfile agent_profile, OctetString agent,
string place_name, ClassNameList class_names,
ASTTAgentSystem agent_sender);

}
```

Figure 7: The ASTT abstract interfaces code par

t

9.3 Receive transferred agent on JADE

The mobile agent that was transferred to the ASTT will be retransferred to JADE, which will then execute the algorithm shown in Figure 2c based on the methods of different classes of JADE that implement serializable classes of Java for the deserialization of received messages, prior to restoring of the agent's state. Agents will be hosted on a dedicated container for receiving agents to help the ASTT to identify them in case it needs to recover them after they are transferred.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

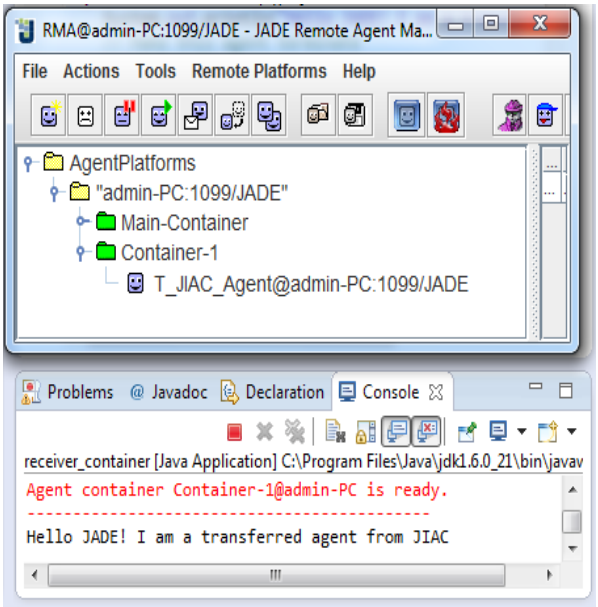


Figure 8 the transformed agent execution result in JADE

10. CONCLUSION AND PERSPECTIVES

In this paper we have developed the algorithm to transform and transfer and mobile agents between JIAC and JADE, taking the information contained in a JIAC agent and creating its equivalent in JADE. We did that by migrating an agent with its functions and attributes, and its added features and their attributes, from one of the two systems to another MAS, based on our previous solution presented in our previous paper, which takes into consideration the application programming interfaces, the architecture language with which both systems were developed.

The research will not stop at this stage. It can be more developed to handle the following cases: 1) when the agent wants to return to his native platform 2) security management in the transfer of agents between different systems. Another line of research that can be developed, managing decision processing agents i.e. to find answers to questions such as: 1) the agent that voluntary decide to return to native MAS, 2) the system that decide to return the agent to its native MAS, 3) how to prioritize several agents that must transferred instantly, 4) how to optimize the management of resource that are shared among agents of different mobile agent systems.

REFERENCES

[1] X. Feng, "A user-oriented approach for selecting information resource service" Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference, 16-18 April 2010, pp. 550 - 553

[2] A. Espinoza, M. Ortega, C. Fernandez, J. Garbajosa, A. Alvarez "Software-intensive systems interoperability in Smart Grids: A semantic approach" in Industrial Informatics (INDIN), 2011 9th IEEE International Conference, 26-29 July 2011, pp. 739 - 744

[3] N. Ide, J. Pustejovsky, "What Does Interoperability Mean, Anyway? Toward an Operational Definition of Interoperability for Language Technology" in 2nd International Conference on Global Interoperability Language for global resource, January 2010

[4] François B. Vernadat, "Technical, semantic and organizational issues of enterprise interoperability and networking"in the Annual Reviews in Control, Volume 34, Issue 1, April 2010, pp. 139–144

[5] V. Fionda, G. Pirró, "Semantic Flow Networks: Semantic Interoperability in Networks of Ontologies Lecture Notes" in Computer Science Volume 7185, 2012, pp. 64-79

[6] Ke-Qing He,Jian Wang, Peng Liang, "Semantic Interoperability Aggregation in Service Requirements Refinement" in Journal of Computer Science and Technology, November 2010, Volume 25, Issue 6, pp. 1103-1117

[7] A. Grimstrup, R. Gray, D. Kotz, M. Breedy, M. Carvalho, T. Cowin, D. Chacon, J. Barton, Ch. Garrett and M. Hofmann "Toward Interoperability of Mobile-Agent Systems" in the Sixth IEEE International Conference on Mobile Agents, Springer-Verlag October 2002, Barcelona, Spain, pp. 106–120.

[8] D. Milojicic et al. MASIF: The OMG mobile agent system interoperability facility. In Proc. of the Second International Workshop on Mobile Agents, volume 1477 of Lecture Workshops (WAINA), 2012 26th International Conference March 2012 pp. 651 – 656

[9] Notes in Computer Science, pp. 50–67. Springer-Verlag, September 1998.

[10] M.Higashino,K.Takahashi, T. Kawamura, K. Sugahara, "Mobile Agent Migration Based on Code Caching" in Advanced Information Networking and Applications

[11] Alfonso Fuggetta and Gian Pietro Picco and Giovanni Vigna,Understanding Code

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Mobility. IEEE Transactions on Software Engineering, 1998, pp. 342—361

[12] A. SOKLABI, M. BAHAJ, I. CHERTI, "JIAC Systems and JADE Agents Communication" in international IJET, May 2013.

[13] G. Stoian, C. Popirlan "A Proposal for an Enhanced Mobile Agent Architecture (EMA)" in annals of the University of Craiova, Mathematics and Computer Science Series, Volume 37(1), 2010, pp. 71-79

[14] N. Suri, J. Vitek "Mobile Agents" in Computational Complexity , 2012, pp 1880-1893

[15] F. Bellifemine, G. Caire, T. Trucco "JADE PROGRAMMER'S GUIDE" 2010.

[16] B. Hirsch,T. Konnerth, A. Heßler "Merging Agents and Services — the JIAC Agent Platform" in Multi-Agent Programming, 2009, pp. 159-185

[17] M. Lützenberger, T. Küster, T. Konnerth, A. Thiele, N. Masuch, A. Heßler, J. Keiser, M. Burkhardt, S. Kaiser, S. Albayrak "JIAC V: A MAS framework for industrial applications" in 13 Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems" 2013, pp. 1189-1199