JAPL: the JADE Agent Programming Language

Mohamed BAHAJ, Abdellatif SOKLABI

FSTS/ Department of Mathematics and Computer Science, University Hassan I

Settat, Morocco

mohamedbahaj@gmail.com

abd.soklabi@gmail.com

***Abstract*— This article describes JADE Agent Programming Language (JAPL) that allows fast and efficient implementation of intelligent behaviors into mobile agents, based on three logical, FIPA speech acts, and a part of complex procedural script for actions. It integrates the ontologies and defines communication services. Rather than rely on a library of plans, JAPL allows agents to plan from first principles. It also describes how to program the multiple JADE behaviors using JAPL instructions and how to compile JAPL to JAVA classes.**

***Keywords*— Mobile agents, JADE, Agent programming language, agents communication, MAS**

I. INTRODUCTION:

JADE (Java Agent Development Environment) is the system for mobile agents, most used in the world. It is adapted to the rules of FIPA [10] and is programmed as a basic object-oriented programming language Java, but the development of a complex application is not yet clear and it requires a lot of concentration, effort and time [4].

In the literature, many programming languages for mobile agents ware created as: Cougaar [6], MetateM [7], AgentSpeak (L) [8], [15], MadKit [12], 3APL [9], [13], Golog [14], [17] and SWAM [3]. These languages are used in the prototyping phase, in order to provide a high level application design-based to mobile agents, who try to follow the architecture Belief- Desire-Intension [16, 15].

In this paper, we present the programming language of JADE agents: "JADE Agent Programming Language". It was designed to implement many complex applications by assigning high mental level concepts to mobile agents, so they reach a new abstraction level that allows their programming as reactive behaviours instead of programming as meaning. In addition to these features, the agents will have the ability to schedule other tasks in order to accomplish the tasks that have been assigned.

First of all, section 2 presents broad overview of the different items JADE Agent Programming Language. Secondly, section 3 talks about the state of the art. Thirdly, section 4 describes its different features in some detail. In particular, we highlight knowledge representation. The following section contains the programming treatment agent's behaviours. The sixth section finalizes this part with a focus on the service concept, and the last section we wrap up with some conclusions.

II. JADE AGENT PROGRAMMING LANGUAGE OVERVIEW:

JAPL is the extension of ADL (Action Description Language) [18] on the mobile agents of JADE. ADL allows the use of quantifiers and conditional expressions in the description of operators to describe classical planning. In ADL, the focus of the quantifiers is over finite domains and disjunction that are not allowed in the expression of effects, because it would bring non-deterministic actions. ADL is a good representation of formal references in classical planning. JAPL is a computer language that standardizes description of planning problems. Also it is a PDDL (Planning Domain Description Language [15]), which allows it to specify the possible operators, the relationships and environmental constraints, the initial state and the goal states.

JAPL provides open world semantics. It includes four essential elements; plans element, rules, ontologies and services. One of the main features for agents is that they can communicate via services. Taking into consideration the view of the agent about the execution, a service call is handled in the same way as the execution of internal actions. This is possible because the services have the same structure as actions. They obtain; pre-conditions and post-conditions, and the body (which contains the actual code to be executed) is reduced to service calls, which allows us to integrate advanced features such as safety, in JADE. In addition, the programming of complex communication scenarios becomes easier, because all messages are processed in a clearly defined framework. In the following sections we will detail some of JAPL's different regions, namely knowledge representation, the behavior of agents, and communications.

doi:10.4304/jetwi.5.3.272-277

III. STATE OF THE ART:

In the literature there are many programming languages of agents, represent a family of programming languages which allows developers to create high-level abstractions and structures which are necessary for the implementation of mobile agents. It is possible to classify them in two categories, those based on logic as AgentSpeak (L) [15], [8], 3APL [9], [13], Golog [17], MetateM [7] and SWAM [3], and those based on object-oriented programming language Java such as Cougaar, [6] and MadKit [12]. These languages are mostly in the prototype stage, and provide high-level concepts that implement some notion of BDI [16], [1].

AgentSpeak (L) programming language is the most developed programming language, and is an article comparing this language to other languages [15]. ALAS and IndiGolog are the newest programming languages of mobile agents. ALAS is a fast, effective, simple and powerful programming language of reactive agents. It has been designed to support the execution of agents in heterogeneous environments, and allow easy use of features of agents, such as mobility [2]. IndiGolog is a programming language for autonomous agents that detect their environment and plan their tasks. IndiGolog supports the execution of high-level programs, gives programmers the ability to create high-level and non-deterministic programs, tests agent tasks and provide a declarative specification of the domain in calculated situations [5]. However, all these languages are widespread and are not suitable for the distinction of all mobile agents systems. We used the strong point in these programming languages for mobile agents and created a simple and efficient language, while respecting the particularity of JADE.

IV. THE KNOWLEDGE REPRESENTATION:

JAPL has been designed to respond to the need of JADE, for a flexible and dynamic language, that allows for the migration of agents and services at any time and which makes the local information's validity very short. So every programming system that supports dynamic behavior needs to address the issue of synchronization and information sharing. So research in the domain of transaction management tries to find solutions to the issue of synchronization [11].

Our approach, however, is to integrate the idea of uncertainty about the bits of information in the presentation of knowledge to allow programmers to manage incorrect or expired information. We integrated the concept of uncertainty using calculated situations which have three assessed logics. The third truth value is added to the predicate and cannot be evaluated with the information available to a particular agent. Thus, a predicate can be explicitly evaluated as unknown. It is an integral part of language, and the programmer is forced to deal with uncertainty in the development of a new agent. Therefore, JAPL can handle incomplete or incorrect information explicitly.

JAPL allows knowledge bases that are most used in the other languages to be defined. Each object that refers to the language needs to be defined in ontology. JAPL implements strong typing, because the variables flow in classes rather than sets of speech. It should be noted here that the alphabet JAPL consists of variables, functions, symbols, actions, quantifiers, connectors, and punctuation symbols. We also use "?" (for testing), "!" (for realization), "and" (for sequencing) and "1" (for implication). Classes are represented in a tree structure. Each node represents a class with a set of attributes. Classes are defined as follows Fig. 1:

```
(Class object classname)
```

Fig.1: Example of JAPL class representation.

JAPL allows multiple inheritances. The classes inherit all attributes of all ancestors. Therefore, the problem of names conflicts are not posed, since the attribute names have been expanded to include the class structure. In addition to classes, JADE Agent Programming Language can define methods and comparisons. The interpretation of the methods is given by the operational semantics. In practice, methods are coded in JAVA.

Complex actions describe functional capabilities of agents. They in turn can call Java methods, or use JAPL. There are different types of complex actions or invocations of services. They all have the same structure; they consist of three main elements, in addition to the name of the action Fig. 2:

```
(action ActionName pre PreCondition eff Effect Body)
```

Fig. 2: Example of JAPL actions representation.

An action is called using the following code Fig. 3:

```
(call role-interface action-id [variable])
```

Fig. 3: JAPL action call example

V. AGENT BEHAVIOUR:

A. *SimpleBehaviour and Action selection:*

As JADE Agent Programming Language is intended to be interpreted in a BDI-like architecture, it incorporates the notion of achievement agent tasks SimpleBehaviour. The SimpleBehaviours of the agent are implemented as

simple commands that the agent tries to respond to once they are activated. Each agent that carries one SimpleBehaviour starts to run by trying to appropriate recovery actions that replay at this SimpleBehaviour. These actions can be either simple scripts or services that are provided by other agents. For this selection, there is no difference between actions that can be performed locally and services. The final selection is made by comparing the orders listed in SimpleBehaviour with the effects of all the known actions of the agent. Comparisons and formulas are made in order to check their compatibility. If they are compatible, the values of variables of SimpleBehaviour agents are linked to the corresponding variables in the action. After the end of the action, the results are written to the original variables of the agent task's orders are evaluated to ensure that the objective of SimpleBehaviour is actually reached. If not, the agent reformulates the composition of its actions, and tries to reach the goal of SimpleBehaviour with other actions.

B. *Reactive Behaviour:*

JADE Agent Programming Language sets rules that control the execution of reactive behaviors of the agent. More specifically, a rule may give the agent a task, whenever a certain event occurs. Rules are applied simply, consisting of a condition and two actions, one of which is executed when the condition is true and the other when it is false Fig. 4.

```
(rule <name>
(var variables-declaration)
conjunction
(true 1)
(false 0)
)
```

Fig. 4: Example of a JAPL rule.

Precisely, whenever an object is added, deleted or modified in the facts, conditions that correspond to the type of the object in the fact are tested and executed. If the result of the test is unknown, no action is taken. For efficiency reasons, the restriction rules that apply to the types of objects have been designed to make it as simple as possible. Actions can themselves be tasks of a new agent or a call to an agent class.

C. *CompositeBehaviour composition and activities:*

In general, the concept of having beliefs, desires and intentions are translated into knowledge belief bases, tasks of agent and a composition library. This allows us to create some principle principles. All these languages require a library of fully developed composition. Overall execution cycle thus consolidates the internal and external states via conjunction function with one or more compositions, which are partially or fully implemented.

In order to achieve a complete composition, the partial compositions must be ordered consistently. As scheduling can be computationally expensive, the algorithm ensures that the main sequence of actions that depend on each other is satisfied. Actions that are executed in parallel are not checked for consistency. Elements of the composition may take a number of forms, which include actions or services as we will detail in the next section.

The agent task execution is discussed as follows. The locally known elements of the actions composition are compared to the agent task. If it finds one with the pre-conditions satisfied, it runs. If not, the composer tries to find others who can satisfy the prerequisites. If the task agent or certain pre-conditions cannot be met with local composers, a request is sent to the Directory Facilitator (DF). If the action is found in the composer, the Directory Facilitator will execute the action. If not a new action is created and executed Fig. 5.

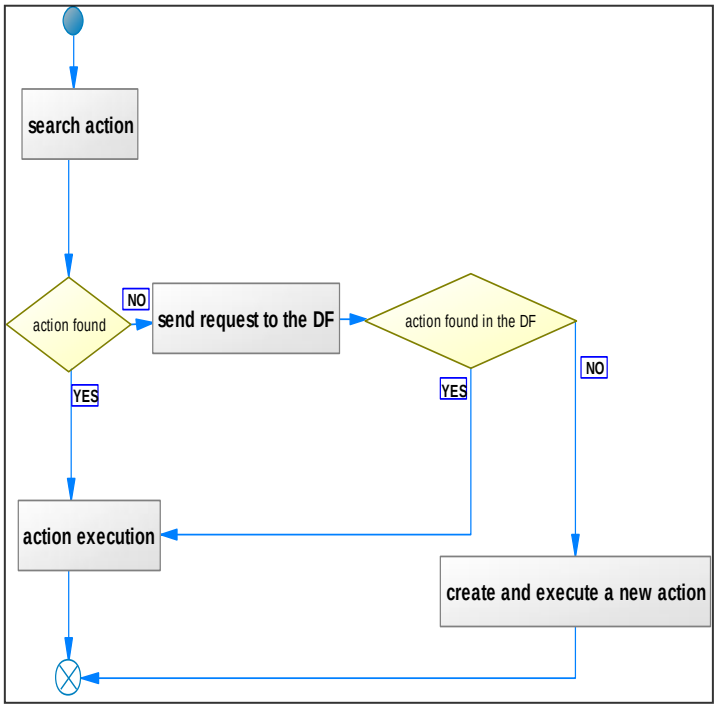

Fig. 5 agent task execution algorithm

Actions may be atomic elements, or they may be scripts. JAPL provides keywords for sequential execution, parallel or conditional, and even the creation of new agent tasks, which then lead to the composition of the new shares.

D. *SequentialBehaviour and ParallelBehaviour:*

JADE Agent Programming Language can define behaviour using sequential blocks that run sequentially, i.e. that all instructions are executed one sequentially. If any

of the statements fail for any reason, the entire block fails. Note that the entire composition does not necessarily fail (Fig. 6).

```
(sequential
(bind ?sender (obj Agent (name
"sender")))
(bind ?receiver (obj Agent (name
"reiceiver")))
(eval (and
(att name ? sender?str)
(not (att name ? receiver?str))
)
)
)
```

Fig. 6 an example of a SequentialBehaviour JAPL

JADE Agent Programming Language offers the possibility of defining Behaviour using parallel blocks running simultaneously, allowing each to be treated separately. If one of the statements must wait for any reason, the other can still be executed without delay. However, the entire execution fails if one of the statements fails, regardless of whether others have been completed or not (Fig. 7).

```
(parallel
(bind ?sender (obj Agent (name
"sender")))
(bind ?receiver (obj Agent (name
"reiceiver")))
(eval (and
(attribut name ? sender?string)
(not (attribut name ? receiver?string))
)
)
)
```

Fig. 7: Example of a ParallelBehaviour JAPL

VI. THE SERVICE CONCEPT:

To allow JADE to be more interoperable with other SMAs, the agents created by JAPL should only communicate using service calls, instead of having an implicit representation of features that can be used by other agents.

All interactions between agents are guided by a generic service. Thus, a service describes an act that the agent performs on behalf of another agent. Services are specified and defined using those conditions and effects. The interaction service always occurs between two agents. An agent must be either the user or the supplier during this operation. The provider is the agent that has some expertise to offer. The other agent in the interaction may act as the service user.

To initiate a service call, the agent must have failed to fulfill a task using only the actions that are available; this includes services that agents provide. If such a situation occurs, the agent sends a request to the DF, which responds with a list of services that could fulfill the task. Then the agent choose a service, and inform the Directory Facilitator, which sends back a return list, but this time a list of agents which provide the requested service.

VII. APPLICATION:

To compile the instructions to create the help in JAPL we had to create a Plugin compilation based on eclipse Plugins. In our case we used the 4.2.1 version of Eclipse. And to build ontology, we call it compilation Plugin. Which we integrate with JAR files JADE (Fig. 8 and Fig. 9).

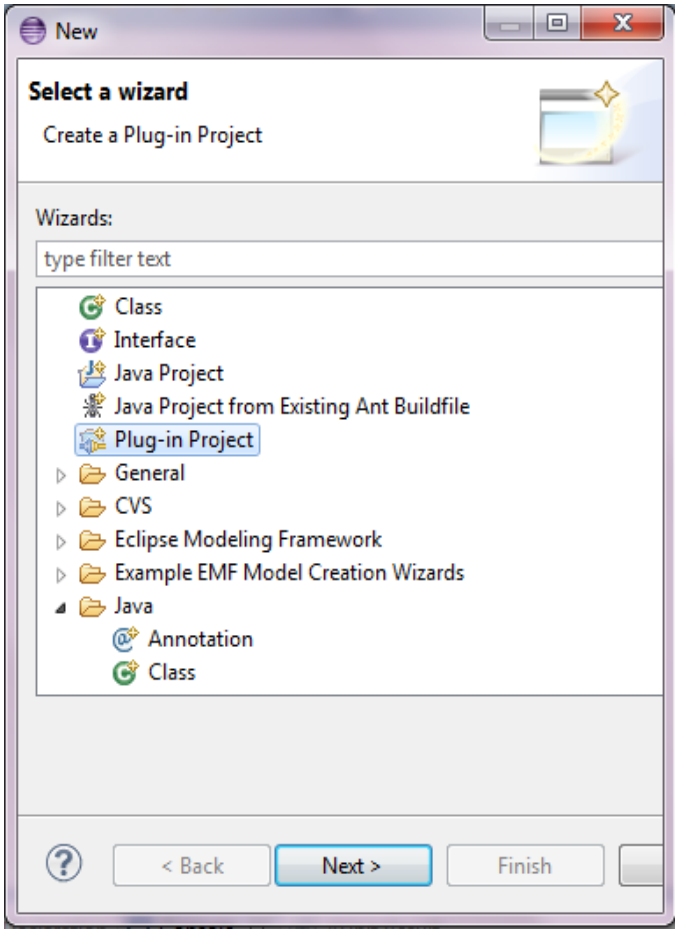

Figure 8: creating a Plugin using Eclipse

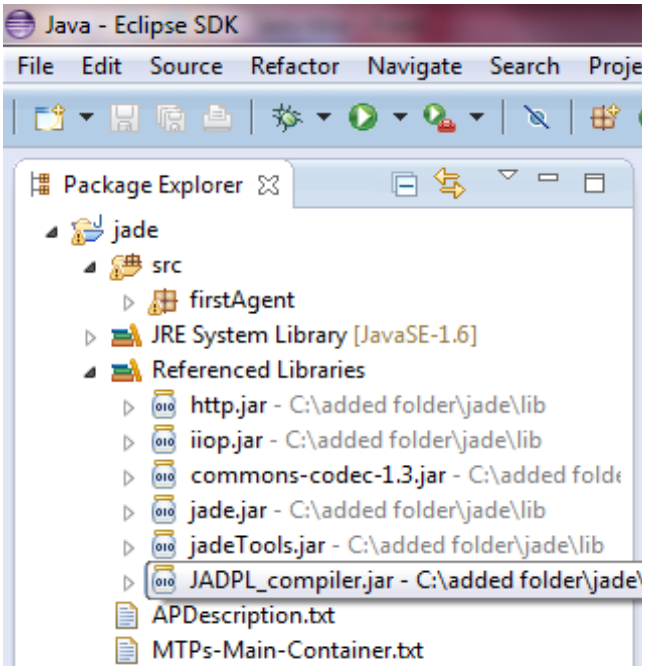

Fig. 9: integration JAPL compilation plugin in the JADE project library.

The compiler takes JAPL input and creates Java classes. These classes must be compiled subsequently using JAVAC. The generated classes implement some JADE interfaces architecture to insure this compilation.

Created classes are:

• OntologyName.java. This class contains the basic structure for classes and their attributes.

• OntologyNameIinterface.java. Java interface contains constants that define types of ontology for each class and attributes.

• OntologyNameMethod.java This class contains the JAVA code for such methods that were written by the programmer without modification.

More details on the use of language JAPL will be available in a users' guide, which soon will be posted on the web soon.

VIII. CONCLUSION:

In this article, we presented JAPL. On the basis of tertiary logic, it provides constructs for describing ontologies, protocols and services, and complex actions for the system of mobile agents JADE. Programmers can use JAPL composition of actions composer. Thus, internal actions and invocations of services are handled transparently to the planning component.

We are confident that JADE agent programming language is likely to be more fruitful than all old languages, in bridging the gap between theory and practice in the development of the JADE mobile agents. Further we are confident that it will push the research in both the pragmatic and theoretical aspects of BDI agents.

REFERENCES

[1] Dennis, M. Fisher, P. Webster, R. Bordini "*Model checking agent programming languages*" in Automated Software Engineering, March 2012, Volume 19, Issue 1, Pages 5-63

[2] D. Mitrovic, M. Ivanovic and M. Vidakovic "*Introducing ALAS: A Novel Agent-Oriented Programming Language* " in NUMERICAL ANALYSIS AND APPLIED MATHEMATICS ICNAAM 2011: International Conference on Numerical Analysis and Applied Mathematics , AIP Conf. Proc. 1389, Pages 861-864, 19–25 September 2011

[3] M. Crasso, C. Mateos, A. Zunino, M. Campo "*SWAM: A logic-based mobile agent programming language for the Semantic Web*" in Expert Systems with Applications Volume 38, Issue 3, March 2011, Pages 1723–1737.

[4] F. Bellifemine, G. Caire, T. Trucco, G. Rimassa "*JADE PROGRAMMER'S GUIDE*" last update: 08-April-2010. JADE 4.0

[5] Giuseppe De Giacomo, Yves Lespérance, Hector J. Levesque, Sebastian Sardina "IndiGolog: a High-Level Programming Language for Embedded Reasoning Agents" Multi-Agent Programming: 2009, Pages 31-72

[6] Helsinger, M. Thome and T. Wright "*Cougaar: A scalabe, distributed multi-agent architecture*" in IEEE SMC04, 2004.

[7] M. Fisher, C. Ghidini and B. Hirsch "*Programming groups of rational agents*" in Fourth International Workshop. Volume 2359 of LNAI. 2004, Pages 16–33.

[8] R. Bordini, J. Hubner et A. Jason: "*a Java Based AgentSpeak Interpreter Used with SACI for Multi-Agent Distribution over the Net*" in 5th edn, 2004.

[9] M. Dastani "3APL Platform" Utrecht University, 2004.

[10] FIPA: Fipa acl message structure specification, 2002.

[11] R. Kotagiri, J. Bailey, P. Busetta "*Transaction oriented computational models for multi-agent systems*" in 13th IEEE International Conference on Tools with Artificial Intelligence, IEEE Press, 2001, Pages 11–17.

[12] O. Gutknecht, J.Ferber "The *madkit agent platform architecture*" in Technical Report, Laboratoire d'Informatique, de Robotique et de Micro électronique de Montpellier, 2000.

[13] Hindriks, K.V., Boer, F.S.D., der Hoek, W.V.and J.J.: Meyer "*Agent programming*" in 3apl. Autonomous Agents and Multi-Agent Systems 2 (1999), Pages 357–401.

[14] G. Giacomo, Y. Lesperance, H. Levesque " *Congolog, a concurrent programming language based on the situation calculus*" Technical report, University of Toronto,1999.

[15] A. Rao: *AgentSpeak(L): "BDI agents speak out in a logical computable language*" in 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96,. Volume 38 of Lecture Notes in Computer Science, 1996, Pages 42–55

[16] S. Rao *"BDI Agents: From Theory to Practice*" Technical Note 56. Australian Artificial Intelligence Institute, April, 1995.

[17] M. Huntbach, N. Jennings and G. Ringwood. "*How agents do it in stream logic programming*" in Proceedingsof the International Conferenceon Multi-Agent Systems, San Francisco, USA, June, 1995.

[18] E. Pednault "*ADL and the state transition model of action*" in logic and computation, 1994, Pages 467-512.

Bahaj Mohamed was born in 1964, in ouezzane, Morocco. He got his PhD in Applied Mathematics, from University of Pau, France, in 1993. He is now working as a Professor at the Department of Mathematics & Computer Sciences, University of Hassan 1er, Faculty of Sciences & Technology of Settat, Morocco. His research interests include pattern recognition, Load Balancing & Controls of mobiles agents, Semantic web & Ontology in MAS.

Soklabi Abdellatif was born in 1985, in El JADIDA, Morocco. He had a license degree in computer engineering in 2009 and a master's degree in computer systems and networks in 2011. Now he is a PhD researcher in mobiles agents and web services in Department of Mathematics & Computer Sciences, University of Hassan 1er, Faculty of Sciences & Technology of Settat, Morocco. His research interests include, Load Balancing & Controls of mobiles agents, Interoperability between different MAS.