

**Subject: Principles of**

**Programming management**

**CSE Department**

**Year/Sem: 3rd Yr. /5th SEM**

**Section: B**

**TOPIC: CASE STUDY ON JADEL**

**Submitted By:**

**Prachi Sawarkar-22**

**A Case Study of**

**the JADEL Programming Language**

JADEL is a novel agent-oriented domain-specific programming language built on top of JADE, the well-known agent platform which provides solid agent technology and several tools for the creation of agents and multi-agent systems. The purpose of JADEL is to make the development of JADE agents and multi-agent systems easier and clearer by means of specific abstractions and a lighter syntax. In order to understand, and properly assess, the actual advantages of using JADEL, a well-known JADE demo that uses JADE specific features like ontologies and interaction protocols has been rewritten in JADEL.

**INTRODUCTION**

JADEL (JADE Language) is a novel programming language that provides agent-oriented abstractions and domain specific expressions to help the development of JADE multi agent systems. JADE (Java Agent Development framework, jade.tilab.com) is a software framework that permits to build complex and distributed multi-agent systems. A wide variety of extensions and APIs are provided with JADE and the documentation of JADE APIs is exhaustive and clear. JADE allows use of agent technology in various areas, such as smart emergency applications and localization.

JADE is currently maintained and in recent years some related projects have been developed. In particular, WADE (Workflows and Agents Development Environment) adds a lightweight workflow engine to JADE agents, thus supporting business process management, and AMUSE (Agent based Multi-User Social Environment) focuses on multi player Android-based online games. These are some of the reasons that make JADE one of the most comprehensive and hence most popular FIPA-compliant agent platform. Although there is a wide and clear documentation of JADE APIs, new developers and students in the field of agents and multi agent systems may have difficulties in approaching JADE.

As a matter of fact, users of JADE have to deal with lots of complementary technical details that are sometimes perceived as difficult and confusing in terms of agent-oriented features. These difficulties are due to the fact that JADE has grown in complexity and now it has a steep learning curve. JADE allows users to make appropriate design choices for their specific application domains, without forcing them to rely on a specific agent model. Such flexibility and adaptability makes it a very powerful instrument that can be applied to several domains. The disadvantage of such an approach is that the agent-oriented features have been used in many different ways, resulting in a loss of transparency of the agent model. Finally, the AOP (Agent-Oriented Programming) paradigm is inherently different from the OOP (Object-Oriented Programming) one. As a matter of fact, agents are characterized by mental states and they exchange specific types of messages, responding to them in a truthful and consistent way. To this extent, agents are viewed as specialization of objects. For this reason, the management of agents and multi-agent systems often requires the use of specific languages, which focus on agent-oriented abstractions and provide particular constructs and structures. In order to match the AOP paradigm, JADE APIs offer facilities written in Java.

**THE JADEL PROGRAMMING LANGUAGE IN BRIEF**

As previously explained the scope of JADEL is to simplify the creation of JADE agents and multi-agent systems and to make the agent model clearly visible, avoiding technical and implementation details as much as possible. JADE APIs consist in a large amount of classes that have various purposes, such as the representation of core abstractions and the management of message passing among agents, from interaction protocols to agent communication languages. Hence, the choice of the primary features, among those provided by JADE Classes , is fundamental in the development of a JADE DSL. In particular, two criteria for the selection of JADEL features are considered, namely

1. the significance of such features in agent creation and life-cycle
2. the FIPA compliance and the help they give to manage message passing.

According to the previously listed criteria, JADEL core abstractions are the agent, the behaviour and the communication ontology. Features of such entities are designed as domain-specific constructs and expressions, whose purpose is to shorten the most repetitive tasks and to focus on the agent life-cycle and behavior action. Constructs are meant to handle events, e.g., the reception of a messages and creation or destruction of agents, while expressions are used, for instance, to create agent behaviors. On the contrary, they represent a tentative of making JADEL code simple and readable, without detracting from its expressiveness. As a matter of fact, the underlying JADE platform and the interoperability with Java given by Xtext and Xtend ensure the possibility of embedding JADE native implementations.

**A. Abstractions**

In terms of syntax, all JADE abstractions are declared in a similar way, namely with a keyword that identifies the abstraction, followed by the name of the entity and its relations with other declared entities. After the declaration, a block of code contains the description of its main features, by means of specific constructs. The first entity that we discuss is the communication ontology. In JADEL they are declared simply by means of the keyword proposition followed by a name. Basic (or atomic) concepts are simple terms that can be used in JADEL, but they do not have to be declared, because they are provided directly by JADEL. Composite concepts are defined by means of other basic or composite concepts. According to JADEL syntax, composite concepts can be declared by using the keyword concept followed by the name of the concept and by a list of parameters. Such parameters identify the other concepts that compose the one that is being declared. Predicates have a similar syntax and their parameters can be basic or composite concepts. Predicates do not state a fact nor describe an entity, but they make relations among concepts, as in the semantics of logic predicates. The second entity is the behaviour. In JADEL, behaviours are used to define actions that can be performed by agents and they can be cyclic or oneshot. The difference between cyclic and oneshot behaviours is that cyclic ones remain in the agent behaviours list for the entire life of an agent, while oneshot behaviours are removed after a single execution. Both of them can be associated with a specific agent class. A special consideration is given to those behaviours that are roles of an interaction protocol. Roles are behaviours defined by means of a FIPA protocol name and the role of the agent in such protocol. In the body of a role, each step of the protocol is handled individually as a particular event caused by the reception of different types of messages. Finally, agents are the most important JADEL entities. In their declarations and setup, they put together all the ontologies and behaviours that will be used in their life cycles, thus

making explicit relations among the various entities. Agents are multitask single-threaded entities that manage their actions by means of an internal scheduler. An agent life cycle is composed of: (i) a start-up phase, during which the agent initializes, (ii) a loop, where actions are performed, and (iii) a 86 final phase of take-down, in which the agent performs cleanup tasks. In JADEL, creation and destruction of an agent are events, so they are managed with specific constructs, whose syntax is coherent with the one of the behaviour events.

**B. Domain-Specific Expressions**

Particular expressions are used to manage events, message passing and domain-specific tasks, such as the behaviors registration in the agent list. Events are managed by means of the keyword on, followed by the type of event. Events can be the creation and the destruction of an agent, the change of state of a participant in an interaction protocol, and the reception of a message. Notably, the JADEL construct on − when − do is used in a behavior body to define an action.

In detail, an action can be simple, i.e., it does not need an event to be triggered. In such a case, only the keyword do is used, followed by a block of code that describes the action. Otherwise, an action can be triggered by an event, and the keyword on specifies such event. The keyword when is optional and it can be used to set conditions on the event nature. Communication among agents consists in message passing, which can be managed with ontologies and/or with interaction protocols. Interaction protocols provide a scenario with, at least, two different roles, initiator and participant, and establish the types of messages exchanged among agents that are in

such roles.

JADEL encourages the usage of such technologies by providing expressions that allow messages to be viewed as special data structures and by means of specific constructs that permit creating and sending messages, and extracting message contents. These expressions and constructs are exactly the same for each type of message, regardless of their content, which can be a string, a sequence of bytes, a concept, a proposition or a predicate.

**THE MEETING SCHEDULER DEMO**

The Meeting Scheduler demo launches two (or more) agents, with their own GUI that consists in a calendar where the user can fix and view his/her appointments, invite known users to an appointment, and receive other users invitations. The agent is responsible for the management of the calendar, so the user does not have to bother about the overlap of two appointments.

As a matter of fact, the agent schedules the appointments correctly by messaging with other agents. The JADE source code of the demo consists in (i) a Meeting Scheduler agent, (ii) a set of classes that manage the GUI, (iii) three behaviors, and (iv) an ontology that defines two composite concepts.

**JADEL Implementation**

The first entity implemented in JADEL is the communication ontology, with the concept Person and Appointment, whose sub-concepts are declared as parameters. The keyword many identifies a list. Declaration of MS Ontology is self contained and it does not need any other classes to define its two concepts, as in JADE.

Then, the agent is declared as a Jadel Gui Agent, an extension of the JADE class GuiAgent that provides some built-in functionalities, e.g., log(String) function, used to manage the logging for the agent. Classes that build the GUI are coded in Java, exactly as in the original demo. It is possible to refer to those classes into the agent body, thanks to the close integration that Xtext and Xtend provide with the underlying JVM. First, some variables are declared in the Meeting Scheduler Agent entity.

Finally, all needed methods of the agent are defined similarly to those in JADE implementation, with some difference regarding domain-specific expressions.

Their declarations indicate the owner agent, protocol name and protocol role.

Roles are finite-state machine behaviors that change state when a new message with a certain per formative is received

Finally, if the proposal is accepted and the responder succeeds in fixing the appointment, a final message that informs the initiator is sent.

The initiator can decide to accept or reject a proposal and, in both cases, it sends a notification message to the responder.

**Quantitative Assessment**

As shown in the previous section, JADEL implementation of the Meeting Scheduler demo is composed by the same entities of the JADE version, but it is characterized by specific syntax for agent-oriented features. Moreover, the operations and methods have the same semantics of those in JADE, but some parts are much shorter than the original ones. Beside the original demo and the JADEL implementation shown above, we obtained a third Java implementation directly by JADEL code generator. The generated code is more redundant than the original JADE demo source code, but it does not introduce significant overhead and the number of messages exchanged by the agents are the same. Hence, there is not a notable performance loss.

The scope of JADEL is mainly to simplify the use of JADE and to make clear and transparent the agent model. In order to evaluate the actual advantages in use of JADEL, we compare the original JADE implementation with our JADEL translation, by using two measures based on Lines of Code (LOC). First, we consider the total number of non-comment and non-blank LOC of the agent, of the ontology, and of the behaviors.

**CONCLUSION**

* Scope and motivations of JADEL development are briefly explained, then main abstractions, domain-specific features and expressions are introduced. The example is presented by showing JADEL source code of the Meeting Scheduler ontology, notable parts of the agent, and all the behaviors.
* A comparison between JADE and JADEL implementations of such a demo is made by counting non-comment and nonblank LOC and by measuring the rate of domain-specific agent-oriented features over the total number of LOC. These indicators are useful in a first approximation to enlighten some JADEL advantages, namely, its lighter syntax and the conciseness in the definition of ontologies.
* Nevertheless, other aspects are difficult to be precisely evaluated, due to their qualitative nature. For instance, some constructs and expressions are meant to reduce the complexity of the framework and to improve readability rather than to reduce the amount of code written.
* In summary, JADEL is shown to be sufficiently expressive to fully reproduce the Meeting Scheduler demo and the best results with respect to JADE are obtained in the ontology implementation.
* Also roles and behaviors are shorter and clearer, due to the frequent use of domain-specific constructs, especially for message passing. Moreover, the entire code of the GUI was reused, thanks to the tight integration of JADEL entities into Java type-system.

**REFERENCES**

* Semanticscholars.com
* Wekipedia