2014 IEEE 26th International Conference on Tools with Artificial Intelligence

An Introduction to the JADEL Programming Language

Federico Bergenti
Dipartimento di Matematica e Informatica
Università degli Studi di Parma
43124 Parma, Italy
Email: federico.bergenti@unipr.it

Abstract

This paper summarizes the current state of development of JADEL, a novel programming language that eases the implementation of agents and multi-agent systems. First, the introduction of a novel agent programming language is motivated and the approach that was used to design JADEL is presented. Then, the characteristic features of JADEL are described by means of a didactic example. The paper is concluded with a short discussion about current and planned developments of JADEL.

1. Introduction

In recent years we witness the rapid growth of the real-world use of agent technology, and we can say that agent technology is quickly becoming a solid tool to develop mission-critical software systems in high-profile settings. Just to cite a notable example, agent technology has been in daily use for service provision and management in Telecom Italia for more than 5 years, serving millions of customers in one of the largest and most penetrating broadband networks in Europe [1].

In order to effectively couple with the inherent issues of such high-profile scenarios, specific tools are needed to assist the developer in the design and implementation of complex functionality, and to promote the effective use of the beneficial features of agent technology. This is the main reason why this paper presents *JADEL (JADE Language)*, a novel programming language that provides explicit and coherent support for *agent-oriented abstractions* and that builds on top of solid agent technology, namely *JADE* (*Java Agent DEvelopment Framework*) (see, e.g., [2]–[5]).

The main objectives of JADEL are: *(i)* to provide developers with high-level abstractions to ensure productivity and software quality; *(ii)* to make sure that the available agent technology is used effectively; and *(iii)* to ensure that the high level of abstraction of the language does not force major methodological or computational trade-offs.

This paper is organized as follows. Section 2 motivates the presented work and it details the objectives of the JADEL project. It also presents the approach that was adopted to achieve the objectives of the project. Section 3 summarizes the characteristic features of JADEL by means of a didactic example that shows how JADEL can be used to implement multi-agent communication using FIPA ACL. Finally, this paper is concluded with a discussion on current and planned developments of JADEL.

2. Why Yet Another Programming Language?

JADE is considered today one of the most appreciated tools to develop agents and multi-agent systems. It has a decennial history of usage in many successful projects both in the academia and in the industry, and it is used to support University courses worldwide. JADE is also the solid base of *WADE* (*Workflows and Agents Development Environment*) (see, e.g., [6]–[9]), an industrial-strength platform for *agent-based business process management*.

Since its initial conception, which dates back to early 1998, JADE is intended primarily to offer support to Java developers. Just a few projects, e.g., the .NET porting of JADE bundled with the source code distribution of JADE itself, tried to loose the tie that couples JADE and Java. This is not surprising and the decision of supporting developers of multi-agent systems with a Java library, rather than with a specific language, is part of the fundamental ideas behind JADE. This approach is also strictly connected with the characteristic features of the development technologies available in the early 2000's. At the time Java was a novel and promising technology, and it was common opinion that it would have changed radically the way software was built. Developers wanted to use it, if nothing else, for their professional growth, and high-profile managers were happy with it because it was the technology that marked the growth of the Web.

Nowadays, we witness a relaxation of the 100% *pure Java* approach of the beginnings because: *(i)* a number of valid alternatives are becoming popular; *(ii)* the minimalistic syntax of Java is perceived as a limitation; and *(iii)* the number of researchers and practitioners advocating the use of *DSLs (Domain Specific Language)* is quickly growing (see, e.g., [10]).

DSLs are intended to give first-class citizenship in the development process to *domain-specific abstractions* by allowing developers to define their own language, specifically tailored for their own needs. DSLs are not meant to replace existing general-purpose languages; rather they are conceived to be integrated into a *host language* that should offer general-purpose features, and that should ensure wide applicability.

For the specific case of using Java as a host language, we already have mature technology for the definition of sophisticated DSLs. Xtext and associated tools [11] are a

1082-3409/14 $31.00 © 2014 IEEE
DOI 10.1109/ICTAI.2014.147

recent technology that allows defining the syntax of a new DSL and, quite easily, implementing a translator to Java for it. The programmer is free to use the new DSL and he/she can immediately have the source code translated into the semantically equivalent set of Java classes. Moreover, the tight integration with the Eclipse IDE ensures that the new DSL can benefit from the appreciated features of one of the most spread and appreciated IDEs. The use of Xtext turns a new DSL into a productivity tool capable of offering to the programmer the right abstractions that he/she needs to address application-specific issues.

The work presented in this paper is an attempt to use the modern technology of DSLs in the scope of mature agent technology, to offer a novel compromise between promising ideas and solid technology. We are not interested in defining a new programming language for agents and multi-agent systems; rather we are interested in enriching a mature programming language with new abstractions. What we do here is by far not simple because we need to take into serious consideration the coherence of the result and the acceptability of our proposal by programmers.

In this sense, JADEL is a new language that builds on top of existing DSL technology and that provides developers of agents and multi-agent systems with agent-oriented abstractions readily available in the programming language.

JADEL is an agent programming language based on Xbase [12], an extensible expression language intended to serve as base for the creation of DSLs. Xbase is not a language on its own; it provides the common features that most, if not all, DSLs require. Each DSL is free to add to Xbase the abstractions that most closely reflect its domain, and JADEL adds to Xbase the core abstractions of agent-oriented programming that have been experienced with JADE for years. Xbase contributes to the overall result with a friendly syntax for all constructs available in Java expressions and with additional features that follow the lesson learned from modern scripting languages.

The use of JADEL offers to the developer the possibility of adopting the right agent-oriented abstractions both in design and in implementation. It also ensures that the gap between agent-based modeling and relative implementation is reduced, thus improving the quality of produced software.

JADEL gives to the developer an agent-oriented shield to the complexity of using JADE. Actually, the maximization of extensibility and reusability made JADE APIs rather articulated and somehow difficult to use. JADEL alleviates this problem by offering a friendly syntax instead of an extensible API.

It is worth noting that we decided not bring into JADEL all the features that JADE offers. Rather, we restricted JADEL to provide only high-level agent-oriented abstractions, and no low-level feature of JADE is accessible. This is not a limitation because, thanks to Xtext and related technologies, the programmer can dig into the Java view of the code and complement it with specific customization that use low-level features, with no manual rework of the generated code.

The tight link with Java is also important for JADEL because of the recent developments of JADE that extend its support (see, e.g., [4], [13]) for mobile scenarios. The porting of JADE to Android [14], and its use in WADE [15] has opened new opportunities for JADE developers. We are also starting a new project, namely *AMUSE* (*Agent-based Multi-User Social Environment*) (see, e.g., [16]–[18]), to use JADE and WADE to support the implementation of mobile social games that can leverage the power of agents to support collaboration (see, e.g., [19], [20]). Needless to say that JADEL has been adapted to such new operative environments with no effort because it can count on generated Java code.

3. JADEL in Brief

JADEL is primarily designed to offer effective concrete syntax for the abstractions that JADE already provides in terms of Java classes. *Agents*, *behaviours*, and *communication ontologies* are the main abstractions that JADEL supports.

Not all the abstractions that JADE currently provides are brought to JADEL because we want JADEL to offer a small set of coherent abstractions that the programmer should perceive at an high level of abstraction. For example, JADEL does not allow the programmer to implement a new codec to be used for sending and receiving FIPA ACL messages. The programmer can choose a codec from a set of available ones, but no new codec can be plugged in using JADEL. This is not a limitation because the developer can dig into the underlying JADE platform to provide his/her codec implementation, but we think that codecs are not an abstraction that fits the JADEL level of abstraction. All in all, JADEL outlines the *agent system level*, as discussed in [21], and it provides a coherent and immediately usable view of it.

The instantiation of agents, i.e., the brought to life of agents and the creation of the multi-agent system, is out of the current scope of JADEL. The programmer needs to dig into JADE, and most probably into activation scripts, to do it.

The choice of leaving the instantiation of the multi-agent system out of the scope of JADEL has been debated intensively, and the current position is that today JADEL is meant to be an *agent* programming language, rather than a *multi-agent* programming language.

Even if we leave aside for a moment all issues related to institutions, organizations, and similar abstractions, the instantiation of a real-world multi-agent system is a serious problem especially for the current target users of JADE, who are interested in robust solutions. In this sense, the instantiation of a multi-agent system should include policies for load balancing and fault tolerance, and it should take into account scalability issues, possibly related to novel technologies like Clouds. WADE is currently one of the most advanced technologies in addressing the problem of instantiating a real-world multi-agent system [1], and we think that JADEL cannot contribute much to it for the moment. This is obviously not a limitation because WADE natively hosts JADE agents, which can be effectively developed using JADEL.

A JADEL system is made of the following set of building blocks:

- A set of interacting agents grouped into *agent classes*—also known as *agent families*.
- A set of behaviours—also known as *tasks*—that agents dynamically adopt to perform their duties and to bring about their goals. They are stateful elements and they are grouped into classes with the same state representation.
- A set of communication ontologies intended to support meaningful communication among agents. Ontologies are purely descriptive elements and they are stateless.

The programmer uses such building blocks to create a JADEL source code and the semantics of the source code is formally expressed in terms of translation rules that turn it into a set of semantically equivalent Java classes. The detailed description of such a translation machinery is out of the scope of this paper and we use here simple informal descriptions to discuss the semantics of JADEL source codes.

Agents, behaviours and ontologies are logically grouped into *packages* to ease the creation of unique names and to avoid name clashes. Packages behave just like ordinary Java packages and they exhibit ordinary semantics for imports. JADEL does not prescribe any rule for placing into files the code of agents, behaviours or ontologies. A single JADEL source code can contain one or more agents, behaviours and ontologies, but it is always associated with just one package.

In the rest of this section we present the major abstractions of JADEL by incrementally building a very simple example of an agent that receives *inform* messages from other agents—not detailed in the example—and that always replies to well-formed messages. Such an agent, commonly known in the JADE community as *Pong*, replies to its counterparts, the *Pings*, to implement a simple *ping-pong* protocol. In this particular incarnation, agents share a common communication ontology, the *PingPong* ontology, and *Pong* responds to messages from *Pings* by informing them of the total number of replies sent.

3.1. Agents

Agents are the executable entities that JADEL provides and they are nothing but JADE agents. An agent is declared in JADEL in terms of several structural features, as follows:

- An entry and an exit procedure, which are implemented as handlers for creation and destruction events.
- The ontologies that the agent uses to communicate. The agent can adopt several ontologies and the order in which ontologies are enumerated in the source code of the agent dictates how name clashes are resolved.
- The peculiar tasks of the agent, expressed as behaviours. JADEL behaviours are mapped to JADE behaviours (see below) and the scheduling policy of active behaviours completely relies on JADE. The order in which behaviours are listed in the source code fixes the order that JADE scheduler uses.

It is worth noting that the declaration of an agent in JADEL is actually the declaration of a class—also known as a family—of agents sharing the features we have just mentioned. With minor abuse of nomenclature we would sometimes say *agent declaration* instead of *agent class declaration* to ease the discourse and because it cannot generate confusion.

The following is an agent declaration in JADEL that specifies how *Pong* agents are structured.

```
package jadel.examples

agents Pong {
  use ontology PingPong

  use active behaviour ReplyWithReceived
  use active behaviour ReplyNotUnderstood

  on create {
    log("Agent " + name + " created")
  }
}
```

This simple declaration creates a new class of agents whose members respond with a confirmation of reception to well-formed messages using the *ReplyWithReceived* behaviour, and they simply respond with a *not understood* message to all other messages by means of the *ReplyNotUnderstood* behaviour.

Agents of class *Pong* handle incoming messages that conform to the *PingPong* ontology only, and they automatically reject messages that do not conform to such an ontology.

The executable code of agents is almost completely placed into behaviours, as detailed below, and only creation and destruction events are handled in the declaration of the agent class. Such a characteristic is exemplified in the code above by the *on create* event handler that simply writes out that the agent has been created using standard JADE logging service.

It is worth noting that, just like for behaviours, the executable code of agents is written as Xbase expressions [12], thus allowing the use of the full power of Java with a simplified and effective concrete syntax. For the specific case of JADEL, we do not value the features that Xbase adds to Java, e.g., closures, because they do not implement agent-oriented abstractions. Needless to say that the programmer can freely use such features in JADEL code.

The class of agents in the example refers to one ontology and to two behaviours, but JADEL allows providing such features in the scope of the declaration itself by means of so called *in-line declarations*. Such declarations are not exemplified in the example and they should be wisely used because they tend to limit code reusability.

A limitation of the current design of JADEL that deserves a discussion is the lack of any sort of inheritance for agent classes. Such a limitation was originally adopted because best practices of JADE programming suggest that the derivation of class *Agent* causes, in the long-term, limitations of the reusability of the features that are placed in the derived class. Moreover, the simplistic approach of mapping the inheritance of agent classes to the inheritance of Java is questionable.

3.2. Behaviours

As stated previously, *behaviours* are the most frequent containers of executable code in JADEL. JADEL behaviours are nothing but JADE behaviours and they are scheduled according to the peculiar policies of JADE. Actually, JADE schedules the active behaviours of an agent in order to implement its functionality, and behaviours can be dynamically activated and deactivated to ensure that agents always have the correct set of active behaviours.

Currently JADEL offers the following three types of behaviour:

- *One shot* behaviours: they are executed only once and they are deactivated just after the execution. They can be explicitly reactivated.
- *Cyclic* behaviours: they are executed an unlimited number of times and they need to be explicitly deactivated.
- *FSM (Finite State Machine)* behaviours: they implement a FSM with an initial state and many termination states.

A notable characteristics of JADEL behaviours, directly inherited from JADE behaviours, is that they are stateful entities, i.e., they carry a runtime state intended to maintain the current state of execution of a task. Because of this peculiarity, JADEL behaviours are better expressed in terms of *classes of behaviours*, just like we do for agents. With minor abuse of nomenclature we would sometimes say *behaviour declaration* instead of *behaviour class declaration* to ease the discourse and because it cannot generate confusion.

The following is the *ReplyWithReceived* behaviour that we need for *Pong* agents.

```
package jadel.examples

cyclic behaviours ReplyWithReceived {
  use ontology PingPong

  on message msg
  when
    performative = INFORM and
    ontology = PingPong and
    content is alive
  do {
    send message {
      ontology = PingPong
      performative = INFORM
      receivers = { msg.sender }
      content = received(counter++)
    }
  }

  int counter = 1
}
```

In this example we declare the class of cyclic behaviours named *ReplyWithReceived*. Such behaviours are used when a new message is received and it matches the specified syntactic criterion in the *when* clause. Such a criterion states that the behaviour is interested in *inform* messages whose content is a single proposition, *alive*, defined in the *PingPong* ontology.

When a message that matches the selection criterion is received, the intended behaviour of the agent is to send a reply message to the sender of the selected message. Such an *inform* reply message contains a proposition *received(counter)* defined in the *PingPong* ontology and used to embed in the message the current value of the counter of sent messages.

The *Pong* agent is also equipped with a lower priority behaviour that traps all messages that the *ReplyWithReceived* behaviour did not catch and it replies with a *not understood* message, as follows.

```
package jadel.examples

cyclic behaviours ReplyNotUnderstood {
  on message msg
  do {
    send message {
      performative = NOT_UNDERSTOOD
      receivers = { msg.sender }
      content = "(" + msg + ")"
    }
  }
}
```

Interested readers should refer to JADE documentation (available from the official Web site [5]) for an in-depth explanation of the three types of behaviours and for a discussion on how JADE schedules active behaviours.

3.3. Ontologies

The *communication ontologies* that JADEL provides are exactly the same ontologies that JADE supports, and they are closely related to specific choices of the FIPA ACL. JADEL provides them primarily for the following objectives:

- To structure the content of outgoing messages.
- To enable basic syntax checks on incoming messages before passing them to active behaviours.
- To dispatch messages to behaviours on the basis of their content.

An ontology in JADEL is declared as a set of structural features that enable knowledge representation in terms of a propositional language. In particular, JADEL allows the declaration of *propositions*, *predicates*, *concepts* and *agent actions*. Concepts are either structured entities or atomic entities, e.g., integers and strings, and they can be used as arguments of predicates. Agent actions are structured entities that can be used to talk about the actions of an agent in the scope of a message.

The *PingPong* ontology can be declared as follows.

```
package jadel.examples

ontology PingPong {
  proposition alive

  predicate received {
    int counter
  }
}
```

The *Pong* agent developed in this section uses the *PingPong* ontology for two types of messages: *(i) inform* messages that,

once received, tells the *Pong* agent that the sender of the message is still alive; and *(ii) inform* messages that the *Pong* agent uses to reply to messages of the previous type. The second type of messages embeds the current value of the counter of sent messages in every message.

It is worth noting that an ontology is used in JADEL to enrich the language with application-specific features. This is the case, e.g., of the *PingPong* ontology, which provides the declaration of the *alive* proposition and of the *received* predicate used in the source code of the *ReplyWithReceived* behaviour.

4. Conclusion

JADEL is meant to ensure that the developer could use the right agent-oriented abstractions both in the design and in the implementation of a system. It is also conceived to ensure that the gap between agent-based modeling and relative implementation is reduced, thus improving the overall quality of results.

The design of JADEL refrained from the simplistic approach of creating a brand new language from scratch. Other authors that developed interesting approaches to the use of DSLs for agent and multi-agent programming (see, e.g., [22]) decided to design a completely new language, rather than enriching an existing language like we do in JADEL. These two approaches are complementary and we cannot yet identify the situations, if any, where one could outperform the other. Similarly, we cannot yet discriminate when the imperative approach of JADEL may result convenient against traditional agent-based languages like AgentSpeak that today can count on solid tools like Jason [23].

In the early experimentations on JADEL that we performed with a class of senior bachelor students and a two expert developers, JADEL was mainly appreciated for the agent-oriented shield that helps managing the complexity of JADE. Even if we still think that JADE is not difficult to use and that the extensive documentation together with the active mailing list make it affordable, we must face the fact that the very advanced use of Java to promote reusability tends to (over-)complicate tasks that programmers perceive as easy. JADEL helps in shielding such a complexity by means of simple abstractions, and the simple comparison of the lengths of the semantically equivalent JADEL and JADE source codes in this paper should clarify this point.

JADEL has not been released to the public yet and the first public release will be available from the JADE Web site [5] as a JADE add-on.

References

[1] F. Bergenti, G. Caire, and D. Gotta, "Large-scale network and service management with WANTS," in *Industrial Agents: Emerging Applications of Software Agents in Industry*, P. Leitão and S. Karnouskos, Eds. Elsevier, 2014.

[2] F. Bellifemine, G. Caire, and D. Greenwood, *Developing multi-agent systems with JADE*. Wiley Series in Agent Technology, 2007.

[3] F. Bellifemine, A. Poggi, and G. Rimassa, "Developing multi-agent systems with a FIPA-compliant agent framework," *Software: Practice & Experience*, vol. 31, pp. 103–128, 2001.

[4] F. Bergenti, A. Poggi, B. Burg, and G. Caire, "Deploying FIPA-compliant systems on handheld devices," *IEEE Internet Computing*, vol. 5, no. 4, pp. 20–25, 2001.

[5] JADE (Java Agent DEvelopment framework) web site. [Online]. Available: http://jade.tilab.com

[6] M. Banzi, G. Caire, and D. Gotta, "WADE: A software platform to develop mission critical, applications exploiting agents and workflows," in *Procs. Int'l Joint Conf. Autonomous Agents and Multi-Agent Systems*, 2008, pp. 29–36.

[7] G. Caire, E. Quarantotto, M. Porta, and G. Sacchi, "WOLF: An Eclipse plug-in for WADE," in *Procs. IEEE Int'l Workshops Enabling Technologies: Infrastructures for Collaborative Enterprises*, 2008.

[8] L. Trione, D. Long, D. Gotta, and G. Sacchi, "Wizard, WeMash, WADE: Unleash the power of collective intelligence," in *Procs. Int'l Joint Conf. Autonomous Agents and Multiagent Systems*, 2009, pp. 1342–1349.

[9] WADE (Workflows and Agents Development Environment) web site. [Online]. Available: http://jade.tilab.com/wade

[10] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Computing Surveys*, vol. 37, no. 4, pp. 316–344, 2005.

[11] Xtext web site. [Online]. Available: http://www.eclipse.org/Xtext

[12] S. Efftinge, M. Eysholdt, J. Köhnlein, S. Zarnekow, W. Hasselbrig, R. von Massow, and M. Hanus, "Xbase: Implementing domain-specific languages for Java," in *Procs. Int'l Conf. Generative Programming and Component Engineering*, 2012, pp. 112–121.

[13] F. Bergenti and A. Poggi, "Ubiquitous information agents," *Int'l J. Cooperative Information Systems*, vol. 11, no. 34, pp. 231–244, 2002.

[14] F. Bergenti, G. Caire, and D. Gotta, "Agents on the move: JADE for Android devices," in *Procs. Workshop From Objects to Agents*, 2014.

[15] F. Bergenti, G. Caire, and D. Gotta, "Interactive workflows with WADE," in *Procs. IEEE Int'l Conf. Enabling Technologies: Infrastructures for Collaborative Enterprises*, 2012, pp. 10–15.

[16] F. Bergenti, G. Caire, and D. Gotta, "Agent-based social gaming with AMUSE," in *Procs. 5^{th} Int'l Conf. Ambient Systems, Networks and Technologies (ANT 2014) and 4^{th} Int'l Conf. Sustainable Energy Information Technology (SEIT 2014)*, ser. Procedia Computer Science, vol. 32, 2014, pp. 914–919.

[17] F. Bergenti, G. Caire, and D. Gotta, "An overview of the AMUSE social gaming platform," in *Procs. Workshop From Objects to Agents*, 2013, pp. 85–90.

[18] AMUSE (Agent-based Multi-User Social Environment) web site. [Online]. Available: http://jade.tilab.com/amuse

[19] F. Bergenti, A. Poggi, and M. Somacher, "A collaborative platform for fixed and mobile networks," *Communications of the ACM*, vol. 45, no. 11, pp. 39–44, 2002.

[20] F. Bergenti and A. Poggi, "Agent-based approach to manage negotiation protocols in flexible CSCW systems," in *Procs. 4^{th} Int'l Conf. Autonomous Agents*, 2000, pp. 267–268.

[21] F. Bergenti and M. N. Huhns, *On the use of agents as components of software systems*. Kluwer Academic Publishing, 2004, ch. 1, pp. 19–32.

[22] A. Ricci and A. Santi, "From actors and concurrent objects to agent-oriented programming in simpAL," *Concurrent Objects and Beyond*, 2013.

[23] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley-Blackwell, 2007.