Rule Engines and Agent-Based Systems

Chapter · January 2008

DOI: 10.4018/9781599048499.ch206

2 authors:

Agostino Poggi
Università di Parma
210 PUBLICATIONS **5,701** CITATIONS
SEE PROFILE

Michele Tomaiuolo
Università di Parma
108 PUBLICATIONS **745** CITATIONS
SEE PROFILE

Rule Engines and Agent-Based Systems

Agostino Poggi, *poggi@ce.unipr.it*
Michele Tomaiuolo*, *tomamic@ce.unipr.it*

Dipartimento di Ingegneria dell'Informazione,
Università di Parma, Italy

INTRODUCTION

Expert systems are successfully applied to a number of domains. Often built on generic *rule-based systems*, they can also exploit optimized algorithms.

On the other side, being based on loosely coupled components and peer to peer infrastructures for asynchronous messaging, *multi-agent systems* allow *code mobility*, adaptability, easy of deployment and reconfiguration, thus fitting distributed and dynamic environments. Also, they have good support for domain specific *ontologies*, an important feature when modelling human experts' knowledge.

The possibility of obtaining the best features of both technologies is concretely demonstrated by the integration of *JBoss Rules*, a rule engine efficiently implementing the *Rete-OO algorithm*, into *JADE*, a *FIPA*-compliant *multi-agent system*.

BACKGROUND

Rule engines

The advantages of *rule-based systems* over procedural programming environments are well recognized and widely exploited, above all in the context of business applications. Working with rules helps keeping the logic separated from the application code: it can be modified by non-developers and, being centralized in one point, it can be analyzed and validated. *Rule engines* are often well optimized, being able to efficiently reduce the number of rules to match against the updated knowledge base.

Rule-based systems can also be augmented with ideas and techniques developed in other research fields, leading for example to fuzzy rule-based systems, which exploit fuzzy logic to deal with imprecision and uncertainty about the knowledge base. Moreover, sometimes these systems are coupled with genetic algorithms and evolutionary programming to generate complex classifiers.

One of the most notable application of rule-based systems are *expert systems*, where the rule set is a representation of an expert's knowledge. In such systems, the AI (Artificial Intelligence) is supposed to perform in a similar manner to the expert, when exposed to the same data.

Among the different mechanisms to implement a rule-engine, *Rete algorithm* (Forgy, 1982) has gained more and more popularity, mainly thanks to the high degree of optimization that can be obtained. At NASA Johnson Space Center, *Rete algorithm* was implemented in a whole generation of rule engines. OPS5 was soon replaced by its descendant, ART, and in 1984 by the more famous CLIPS.

Nowadays, one of the most widespread engines implementing *Rete* is *Jess* (Friedman-Hill, 2000), at first developed as a Java port of CLIPS at Sandia National

Laboratories in late 1990s. *Jess* has also been widely adopted by the agent community to realize rule-based agent systems (Cardoso, 2007).

A different yet promising rule-engine is *JBoss Rules* (Proctor, Neale, Frandsen, Griffith & Tirelli, 2007), formerly *Drools*. It is a quite new, but already well known, freeware tool implementing so-called *Rete-OO algorithm*.

Its open-source availability is a clear advantage over *Jess*, but an even greater advantage is due to the implementation of a particular adaptation of the *Rete algorithm* for the object-oriented world, rather than a literal one. This way, the burden of integrating the rule-engine and application rules with existing external objects is greatly reduced. In fact, *JBoss Rules* uses plain Java objects to represent rules and facts, which can be modified through their public methods and properties. Rules can be specified through an appropriate syntax, or through xml structures, and their conditions and consequences can be expressed using different scripting languages, as Python, Groovy and Java. Instead *Jess* only accepts rules written in the CLIPS language, thus requiring developers to learn a new Lisp-like language and deploy additional efforts to adapt it to their object-oriented development environment.

```
package org.drools.examples

import org.drools.examples.HelloWorldExample.Message;

rule "Hello World"
when
     m : Message( status == Message.HELLO, message : message )
then
     System.out.println( message );
     m.setMessage( "Goodbye cruel world" );
     m.setStatus( Message.GOODBYE );
     update( m );
end

rule "GoodBye"
no-loop true
when
     m : Message( status == Message.GOODBYE, message : message )
then
     System.out.println( message );
end
```

Figure 1. Basic example from the JBoss Rules handbook

Agent-based systems

Multi-agent systems (MAS) show some complementary features which can be useful in many rule-based application, above all asynchronous interaction protocols and semantic languages. In multi-agent systems, in fact, many intelligent agents interact with each other. The agents are considered to be autonomous entities, and their interactions can be either cooperative or selfish (i.e. they can share a common goal, as in a production line, or they can pursue their own interests, as in an open marketplace).

The *Foundation for Intelligent Physical Agents* (FIPA, 2002) develops open specifications, to support interoperability among agents and agent-based applications. Specifications for infrastructures include a communication language for agents, services for agents, and they anticipate the management of domain-specific *ontologies*. A set of application domains is also specified, including personal assistance for travels, network

management, electronic commerce, distribution of audio-visual media. At the core of FIPA model there's the communication among agents; in particular it describes how the agents can exchange semantically-meaningful messages with the aim of completing activities required by the overall application.

Various implementations of FIPA-compliant platforms exist (FIPA implementations, 2003). Among them, *JADE* (Bellifemine, Caire, Poggi & Rimassa, 2003) has gained popularity during the years, while more and more core functionalities and third-party plug-ins were being developed. Currently it supports most of the infrastructure related *FIPA* specifications, like transport protocols, message encoding, and white and yellow pages agents. Moreover, it has various tools that ease agent debugging and management.

The possibility of using rules to realize agent systems seems to be promising. On the one hand, rules have been shown suitable to define abstract and real agent architectures and have been used for realizing so-called "*rule-based agents*", that is, agents whose behaviour and/or knowledge is expressed by means of rules (Shoham, 1993) (Rao, 1996) (Hindriks, de Boer, van der Hoek & Meyer, 1998) (Schroeder & Wagner, 2000). On the other hand, given that rules are easy and suitable means to realize reasoning, learning and knowledge acquisition tasks, rules have been used into so-called "*rule-enhanced agents*", that is, agents whose behaviour is not normally expressed by means of rules, but that use a rule engine as an additional component to perform specific reasoning, learning or knowledge acquisition tasks (Gutknecht, Ferber & Michel, 2000) (Katz, 2002). Both the approaches have some advantages and disadvantages. *Rule-based agents* provide all the advantages of *rule-based systems* and a uniform way to program them, but their performance is inadequate for some kinds of applications. *Rule-enhanced agents* allow the use of different programming paradigms; therefore, it is possible to use the most appropriate paradigm for the realization of the different tasks both to simplify the development and to satisfy the performance requirements, but there is an additional cost for the management of the integration/synchronization of such heterogeneous tasks. With behaviour-based agents, as in *JADE*, the *rule engine* can be integrated into an agent as a behaviour. This approach can alternatively guarantee the advantages of full *rule-based agents* or the ones of *rule-enhanced agents*. In facts, both procedural and rule-based behaviours can be seamlessly added to each deployed agent, according to the application features and requirements.

Code mobility and security

Mobile code proves useful in many contexts (Fuggetta, Picco & Vigna, 2000), thanks to its ability to overcome network latency, reduce network load, allow asynchronous execution and autonomy, adapt dynamically, operate in heterogeneous environments, provide robust and fault-tolerant behaviours. *Mobile code* technologies vary from applets and other dynamic code downloading mechanisms, to full mobile agent systems, adhering to models as code on demand, remote evaluation, mobile agents. When a rule engine is integrated into a multi-agent system, two different cases are possible: asking a remote agent to execute a task, or to apply a new rule to its knowledge base. While mobile rules falls into the class of asynchronous requests with deferred execution, instead mobile tasks fall into the synchronous class. In both cases the moved entity is a

fragment of code, to be interpreted by a scripting engine on the target agent, and not a complete thread of execution.

The different *security threats* that a *mobile code* system could face, and the relevant security countermeasures that could be adopted, should also be analyzed. In (Jansen & Karygiannis, 2000) two different classes of attacks are identified, depending on their target: the ones targeting the executing environment of mobile code, and the ones targeting the code itself. While the fact that mobile code could pose threats to its hosting environment is widely accepted, instead often the possibility to face threats against the hosted code is not taken into consideration. This is certainly due to a lack of effective countermeasures to prevent the hosting environment from stealing data and algorithms from the mobile code, from executing it too slowly to be effective, altering its execution flow, or stopping its execution. Experimental algorithms exist to at least detect "a posteriori" this type of threats, including partial result encapsulation, mutual itinerary recording, itinerary recording with replication and voting, execution tracing. Some algorithms even try to prevent some types of attacks to the code hosted in malicious environments, but their real effectiveness has yet to be proved; these include environmental key generation, computing with encrypted functions, and obfuscated code (sometimes called time limited blackbox). On the other hand, potential threats posed by hosted code include masquerading, denial of service, eavesdropping, and alteration. Available security countermeasures to protect the execution environment against potentially malicious mobile code often rely on algorithms to prevent attacks, like software-based fault isolation, safe code interpretation, authorization and attribute certificates, proof carrying code. Other techniques are focused on detecting attacks to the environment and tracing them to their origin; these include state appraisal, signed code, path histories.

Systems based on Java can leverage on the security means provided by the virtual machine, and extend them as needed. In particular, it is possible to define precise protection domains on the basis of *authorization certificates* (Poggi, Tomaiuolo & Vitaglione, 2004). These certificates, attached to mobile code, list a set of granted permissions and are signed by local resource managers. Access rights can also be delegated to other agents, to allow them to complete the requested tasks or to achieve delegated goals (Somacher, Tomaiuolo & Turci, 2002). Finally, masquerading and alteration threats can be prevented by establishing authenticated, signed and encrypted channels between remote components of the system.

INTEGRATION OF RULES AND AGENTS

Among the different implementations of *rules-enhanced multi-agent systems*, the analysis Drools4JADE (Drools4JADE) can be particularly interesting, as the system resulted from the evaluation of existing technologies in various fields. In fact, the purpose of this project was to not start from scratch, to develop of a totally new agent platform, but instead to build on existing solutions, which already demonstrated to be a sound layer on which more advanced functionalities could be added.

In this case, the chosen agent system is *JADE* (Bellifemine, Caire, Poggi & Rimassa, 2003). Its successful adoption in large international projects, like Agenticities (Poggi, Tomaiuolo & Turci, 2004), openNet and TechNet (Willmott, 2004), proved it to be preferable to other solutions, thanks to its simplicity, flexibility, scalability and

soundness. As already argued, its integration with an open source object-oriented rule engine, as *JBoss Rules*, in many contexts is to be favoured against the more traditional *JADE-Jess* couple (Cardoso, 2007).

FIPA Interface to the Rule Engine

To the rich features of *JBoss Rules*, an agent environment can add above all the support for communications through ACL (Agent Communication Language) messages, typical of *FIPA* agents. Rules can reference ACL messages in both their precondition and consequence fields. Moreover, a complete support to manipulate facts and rules on rules-enhanced agents through ACL messages can be provided.

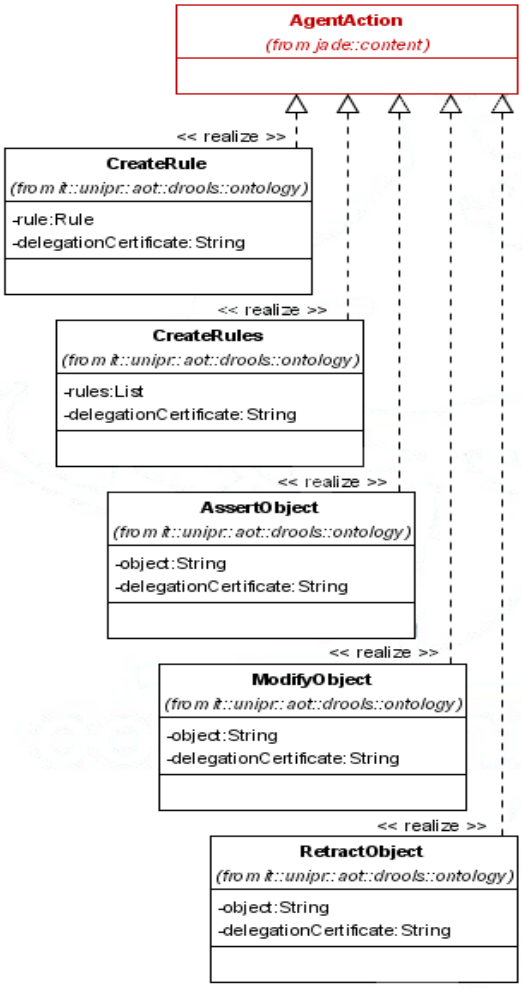

Figure 2. Actions supported by rules-enhanced agents

Inside the *JBoss Rules* environment a rule is represented by an instance of the Rule class: it specifies all the data of the rule itself, including the pre-conditions making the rule valid and the actions to be performed as consequence of the rule. When a rule is scheduled for execution, i.e. all its preconditions are satisfied by asserted facts, the engine creates a new instance of the embedded scripting environment, set the needed variables

inside it and invokes the interpreter to execute the code contained in the consequence section of the rule.

In Drools4JADE, rules-enhanced agents expose a complete API to allow the manipulation of their internal working memory through ACL requests. Their *ontology* defines requests to add rules, assert, modify and retract facts. All these requests must be joined with an *authorization certificate*. Only authorized agents, i.e. the ones that show a certificate listing all needed permissions, can perform requested actions. Moreover, the accepted rules will be confined in a specific protection domain, instantiated according to their own *authorization certificate*.

Security issues

Mobility of rules and code among agents paves the way for really adaptive applications, but it cannot be fully exploited if security issues aren't properly addressed. The security means implemented in Drools4JADE greatly benefit from the existing infrastructure provided by the underlying Java platform and by *JADE*. The security model of *JADE* deals with traditional user-centric concepts, as principals, resources and permissions. Moreover it provides means to allow delegation of access rights among agents, and the implementation of precise protection domains, by means of authorization certificates.

In the security framework of *JADE*, a principal represents any entity whose identity can be authenticated. Principals are bound to single persons, departments, companies or any other organizational entity. Also single agents are bound to a principal; with respect to his own agents, a user constitutes a parent principal, thus allowing to grant particular permissions to all agents launched by a single user.

Resources that *JADE* security model cares for include those already provided by security Java model (i.e. file system, network connections, environment variables, database connections). Resources typical of multi-agent systems, to be protected against unauthorized accesses, include agents themselves and their executing environment.

A permission represents the capability to perform actions on system resources. To take a decision while trying to access a resource, access control functions compare permissions granted to the principal with permissions required to execute the action; access is allowed if all required permissions are owned.

When an agent is requested to accept a new rule or task, a first access protection involves authenticating the requester and checking the authorization to perform the action; i.e.: can the agent really add a new rule, or submit a task, to be performed on its behalf? To perform these tasks, the requester needs particular permissions.

Moreover, to exploit the full power of task delegation and rule mobility, the target agent should be able to restrict the set of resources made accessible to mobile code. Agents should be provided means to delegate not only tasks, but even access rights needed to perform those tasks. This is exactly what is made possible through the security package of JADE, where distributed security policies can be checked and enforced on the basis of signed *authorization certificates*.

In this kind of systems, every requested action can be accompanied with a certificate, signed by a local resource manager, listing the permissions granted to the requester. Permissions can be obtained directly from a policy file, or through a delegation

process. Through this process, an agent can further delegate a set of permissions to another agent, if it can prove the possession of those permissions.

The final set of permissions received through the request message, can finally be used by the servant agent to create a new protection domain to wrap the mobile code during its execution, protecting the access to system, as well as application, resources.

FUTURE TRENDS

A system architecture founded on *rule engines* and *multi-agent systems* is a good starting point to build a fully distributed environment, where the distributed knowledge can include both data and code. For example it can be used to realize systems for distributed signals and alarms handling, network management etc.

The development of advanced grid features, as transparent and reconfigurable functions for dynamic load balancing, distribution of facts and rules among remote engines, failure detection and recovery, could add even greater value to the system, paving the way for the development of distributed computing environments founded on networks of *FIPA* agents and platforms.

CONCLUSION

The integration of an object-oriented *rule engine* and a scripting engine into an agent development framework can provide many advantages. The resulting system joins the soundness of a platform for distributed *multi-agent systems*, with the expressive power of rules and the ability to adapt to changing conditions granted by *mobile code*.

Of course, the development of real world applications poses serious security requirements, which can be faced by means of detailed security policies and delegation of *authorizations* through signed *certificates*. Application areas include, but certainly are not limited to, e-learning, e-business, service-composition, network management.

REFERENCES

Bellifemine, F., Caire, G., Poggi, A., Rimassa, G. (2003). JADE A White Paper. *Telecom Italia EXP magazine* Vol 3, No 3 September 2003.

Cardoso, H.L. (2007). Integrating JADE and Jess. JADE Web site: http://jade.tilab.com/doc/tutorials/jade-jess/jade_jess.html.

Eberhart. (2002). OntoAgent: A Platform for the Declarative Specification of Agents, In *Proc. of ISWC 2002*, Cagliari, Italy.

FIPA. (2007). FIPA Abstract Architecture Specification. FIPA Web site: http://www.fipa.org/specs/fipa00001/SC00001L.html.

FIPA. (2003). Publicly Available Implementations of FIPA Specifications. FIPA Web site: http://www.fipa.org/resources/livesystems.html.

Forgy, C. L. (1982). Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem, *Artificial Intelligence* 19(1), pp. 17-37.

Friedman-Hill, E.J. (2000). Jess, the Java Expert System Shell. Sandia National Laboratories Web site: http://herzberg.ca.sandia.gov/jess.

Fuggetta, A., Picco, G.P., Vigna, G. (1998). Understanding code mobility, *IEEE Transaction on Software Engineering* 24 (5):342–362.

Gutknecht, O., Ferber, J., Michel, F. (2001). Integrating tools and infrastructures for generic multi-agent systems. In *Proc. of the Fifth Int. Conf. on Autonomous Agents*. Montreal, Canada.

Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.C. (1998). Control Structures of Rule-Based Agent Languages. *Proc. ATAL-98*, Paris, France.

Jansen, W., Karygiannis, T. (2000). Mobile agent security. *NIST Special Publication 800-19*.

Katz, E.P. (2002). A Multiple Rule Engine-Based Agent Control Architecture, *Technical Report HPL-2001-283*. HP Laboratories Palo Alto.

Poggi, A., Tomaiuolo, M., Vitaglione, G. 2004. A Security Infrastructure for Trust Management in Multi-agent Systems. *Trusting Agents for Trusting Electronic Societies* 2004: 162-179.

Poggi, A., Tomaiuolo, M., Turci, P. (2004). Using Agent Platforms for Service Composition. ICEIS (4) 2004: 98-105

Proctor, M., Neale, M., Frandsen, M., Griffith, S.Jr., Tirelli, E. (2007). JBoss Rules User Guide. JBoss Web site: http://labs.jboss.com/jbossrules/docs.

Rao, A.S. (1996). AgentSpeak(L): BDI Agent Speak Out in a Logical Computable Language. *Agents Breaking Away*: 42-55.

Schroeder, M., Wagner, G. (2000). Vivid agents: Theory, architecture, and applications. *Int. Journal for Applied Artificial Intelligence*, 14(7): 645-676.

Shoham, Y. (1993) Agent-oriented programming. *Artificial Intelligence*, 60(1): 51-92.

Somacher, M., Tomaiuolo, M., Turci, P. (2002). Goal Delegation in Multiagent System. *AIIA 2002*. Siena, Italy.

Willmott, S. (2004). Deploying Intelligent Systems on a Global Scale. *IEEE Intelligent Systems* Vol. 19(5), September/October 2004: 71-73.

TERMS AND DEFINITIONS

Rule-based system: created using a set of assertions, which collectively form the "working memory", a database which maintains data about current state or knowledge, a set of rules, specifying how to act on the assertion set, and a rule-engine or interpreter. Basically, rule-based systems can consisting of little more than a set of if-then statements, but provide the basis for so-called "expert systems".

Production system (or production rule system): a rule-based system whose rules (termed productions) consist of two parts: a sensory precondition (or "if" statement) and an action (or "then"). If a production's precondition (left-hand side or LHS) matches the current state of the world, then the production is said to be triggered. If a production's action is executed, it is said to have fired. The rule interpreter must provide a mechanism for prioritizing productions when more than one is triggered. Rule interpreters generally execute a forward chaining algorithm for selecting productions to execute.

Expert system: encodes the knowledge of an expert into the rule set of a rule-based system. When exposed to the same data, the expert system AI will perform in a similar manner to the expert.

Software agent: a software entity being able to act with a certain degree of autonomy, in order to accomplish tasks on behalf of its user. While objects are defined in terms of methods and attributes, agents are defined in terms of their behaviours. Usually agents show persistence, autonomy, social ability, reactivity.

Multi-agent system: a software system based on the interaction of several agents. Such agents could not have all data or all resources needed to achieve an objective and need to collaborate with other agents. In this case, data is decentralized and execution is asynchronous. Earlier, related fields include Distributed Artificial Intelligence (DAI) and distributed problem solving (DPS).

Ontology: an explicit specification of a conceptualization, formally describing the entities involved in a particular domain and the relationships among them.

Authorization certificate: a digital document that describes a permission from the issuer to use a service or a resource that the issuer controls or has access to use. Usually it is signed by means of a public key algorithm. The permission in some case can also be delegated.