WADE – An Open Source Platform for Workflows and Agents

G. Caire, E. Quarantotto, G. Sacchi

Telecom Italia
Via Reiss Romoli 274
10148 Torino - Italy

Categories and Subject Descriptors
I.2.11 {**Artificial Intelligence**]: Distributed Artificial Intelligence - *Multiagent systems*; C.2.4 {**Computer Communication Systems**]: Distributed systems; D.2.11 [**Software Engineering**]: Software architecture

General Terms
Management, Performance, Languages.

Keywords
Software Agent, workflow, JADE, Open Source, XPDL, Scalability, Flexibility.

1. INTRODUCTION
WADE is the main evolution of JADE and adds to it the ability to define system logics according to the workflow metaphor.

WADE is not just an add-on, but a complete platform (built on top of JADE) providing advanced administration and fault tolerance mechanisms and enabling a group of agents to cooperatively execute complex tasks defined as workflow.

Nowadays workflows are mostly adopted in BPM (Business Process Management) environments where they are used to represent business processes and orchestrate existing systems typically (but not necessarily) accessible by means of Web Services-based interfaces.

The main challenge in WADE is to bring the workflow approach from the business process level to the level of system internal logics. That is, even if in principle it could be used for that purpose too, WADE does not target high level orchestration of services provided by different systems, but the implementation of the internal behaviour of each single system. Each agent embeds a micro-workflow engine and a complex process can be carried out by a set of cooperating agents each one executing a piece of the process.

The approach followed by WADE is to provide a workflow view on top of a normal Java class. That is a workflow is implemented as a Java class with a well defined structure. A key element in this approach is WOLF (WOrkflow LiFe cycle management environment), the graphical development environment for WADE based applications.

The presentation will provide an overview of the WADE platform, will highlight its most distinguishing features such as workflow inheritance, support for web services and delegation.

2. WADE OVERVIEW
WADE (Workflow and Agent Development Environment) is a domain independent platform, built on top of JADE 5, an open source middleware for the development of distributed applications based on the agent-oriented paradigm. The distribution of JADE includes a runtime environment, a library of classes that programmers can use to develop their application and some graphical tools for administration and monitoring purposes.

Each running instance of the JADE runtime environment is called Container and a set of containers is called Platform. In a JADE Platform a single special Main Container must always be active and the other containers register with it at startup.

One or more application agents can be started into a Container. The actual job of an Agent is to perform some tasks assigned to it. In JADE, a "Behavior" represents a task to be performed by an Agent and it is implemented as an object of a class that extends the class `Behaviour` of the JADE library.

An Agent to perform its tasks may need to communicate with other Agents in the Platform. JADE provides the agents with the ability to communicate. The communication model adopted is the "Asynchronous Message Passing" and the format of the messages is the ACL (Agent Communication Language) defined by FIPA 5.

WADE adds to JADE the support to the workflow execution and a few mechanisms to manage the complexity of the distribution, in terms of administration and fault tolerance. It should be noticed that a WADE-based application may even not use Workflow Engine agents at all and just exploit the administration and fault tolerance features. In that case we end up with a WADE-based application that does not use workflows. On the other side, it is also possible to use the workflow metaphor inside of a Jade platform.

Wade specific components are:

- **BootDaemon processes**: there is a bootDaemon process for each host in the platform and it is in charge of the Containers activation in its local host.
- **Configuration Agent (CFA):** the configuration agent always runs in the Main Container and is responsible for interacting with the boot daemons and controlling the application life cycle.
- **Controller Agents (CA):** there is a controller agent for each container in the platform and they are responsible for supervising activities in the local container and for all the fault tolerance mechanisms provided by WADE.

- **Workflow Engine Agents (WEA):** Workflow Engine Agents embed an instance of the micro workflow engine and therefore they are able to execute workflows.

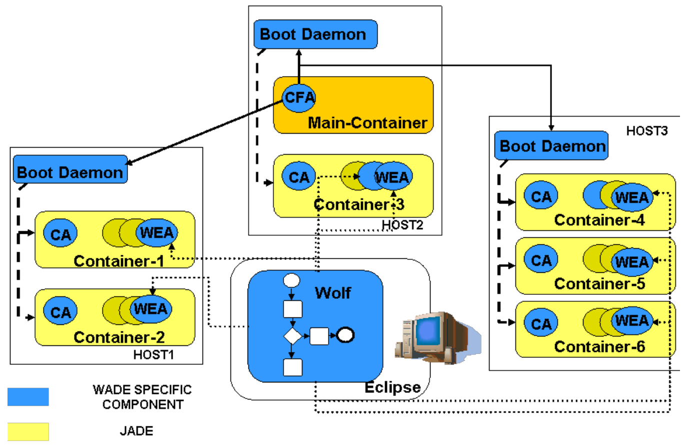


Figure 1 – a WADE platform

Figure 1 shows the topology of a WADE-based application. WADE specific components are highlighted in blue.

As mentioned, workflows are represented in Wade as Java classes, so, in principle, Wade supports "notepad-programming" in the sense that there is not hidden stuff that developers can't control. However, expecially considering that one of the main advantages of the workflow approach is the possibility of representing processes in a friendly graphical form, Wade comes with a development environment called Wolf that facilitates the creation of Wade-based applications. Wolf is an Eclipse plug-in and as a consequence allows Wade developers to exploit the full power of the Eclipse IDE plus additional Wade specific features.

3. Workflow approach

A workflow is a formal definition of a process in terms of activities to be executed, relations between them, criteria that specify the activation and termination and additional information such as the participants, the software tools to be invoked, required inputs and expected outputs and internal data manipulated during the execution.

The key aspect of the workflow metaphor is the fact that the execution steps as well as their sequencing are made explicit. This makes it possible to give a graphical representation of a process defined as a workflow. Such representation is clearly extremely more intuitive with respect to a piece of software code and in general is understandable by domain experts as well as by programmers.

Domain experts can therefore validate system logics directly and not only on documents that most of the time are not perfectly up to date. In some cases they could even contribute to the actual development of the system without the need for any programming skill.

Another important characteristic is that, being the execution steps explicitly identified, the workflow engine (i.e. a system able to automatically execute a process defined as a workflow) can trace them. This makes it possible to create automatic mechanisms to facilitate system monitoring and problem investigation.

Additionally, when processes have to be executed within the scope of a transaction, semi-automatic rollback procedures can be activated in case of unexpected fault. Finally, since workflows are fully self-documented, workflow-based development releases the development team of the burden of keeping documentation aligned each time design choices must be revisited to face implementation details or evolving requirements.

Nowadays the workflow metaphor is mostly used in BPM environments where a workflow represents a business process and orchestrates a number of existing systems typically (but not necessarily) accessible by means of Web Services based interfaces.

The main challenge in WADE is to bring the workflow approach from the business process level to the level of system internal logics. That is, even if it could be used for that purpose too, WADE does not target high level orchestration of services provided by different systems, but the implementation of the internal behaviour of each single system.

First of all it should be noticed that WADE does not include a single powerful workflow engine as the majority of BPM oriented tools. On the contrary WADE provides an extension of the basic Agent class of the JADE library called `WorkflowEngineAgent` that embeds a small and lightweight workflow engine (we talk about "micro-workflow engine"). As a consequence, besides normal JADE behaviours, all Workflow-Engine agents active in a WADE-based multi-agent applications are able to execute workflows represented according to a WADE specific formalism.

The second important point to highlight is that, in order to allow developers to exploit the workflow metaphor to define system internal logics and, at the same time, to give them the same power of a software programming language and a comparable execution efficiency, the WADE workflow representation formalism is based on the Java language. That is, a workflow that can be executed by WADE Workflow-Engine agents is expressed as a Java class with a well defined structure (as it will be explained in the following). As such WADE workflows can be edited, refactored, debugged and in general managed as all Java classes and can include all pieces of code (methods, fields of whatever types, inner classes, references to external classes and so on) needed to implement the process details. In addition, of course, the execution flow they specify can be presented and modified in a friendly, graphical way. More in details WOLF (the development environment for WADE based applications) is an Eclipse plugin and allows developers to work with a graphical view (suitable to manage the process flow) and a code view (the usual Eclipse Java editor suitable to define execution details) that are kept in synch.

Finally it must be noticed that WADE does not impose that all system logics are defined as workflows. Developers are free to exploit the workflow metaphor to describe those tasks for which they think it is appropriate and use normal JADE behaviours (or other purely Java patterns) elsewhere.

As mentioned the approach followed by WADE is to provide a workflow view on top of a normal Java class.

As a consequence, a workflow is implemented as a Java class and no standard workflow definition language is used. However, Wolf adopts the workflow meta-model defined in the XPDL [2], standard specified by the Workflow Management Consortium. The XPDL meta-model has been chosen, because the XPDL language has been conceived as interchange formalism between different systems. WADE supports the import of XPDL files and the adoption of this meta-model facilitates these operations. Moreover, the XPDL meta-model is based on a Finite State Machine computational model that is the same model supported by the WADE agents.

In the XPDL meta-model a process is represented as a workflow, consisting of one or more **activities** that can be thought as tasks to be executed.

In a workflow, the execution entry point is defined, specifying the first activity to be performed; this activity is called **Start Activity.** On the other hand, a workflow must have one or more termination points, named **Final Activities.**

The execution flow is defined by means of **transitions**. A transition is an oriented connection between two activities and may have a condition associated. Regular or exception transitions can be defined. Exception Transitions allow specifying branches that are taken only when an Exception is raised in the source activity.
Excluding the final ones, each activity may have one or more outgoing transitions. When the execution of an activity is terminated, the conditions associated to its outgoing transitions are evaluated. As soon as a condition is verified the corresponding transition is activated and the execution flow proceeds towards the destination activity.

Normally a process execution uses some internal data, for instance, to pass intermediate results between activities and/or for evaluation of conditional expressions. In the XPDL meta-model internal data are modeled by **Data Fields**.

A process can have one or more inputs to be provided and one or more outputs expected at the end of its execution. Inputs and outputs of a process can be formalized in the XPDL meta-model by means of the workflow **Formal Parameters**.

The XPDL meta-model defines some predefined types of activity. The most important ones are:

- **Tool Activity**: a tool activity is an activity that is implemented by means of the invocation of one or more software tools, named **Applications**.
- **Subflow Activity**: a subflow activity is an activity that requires the execution of another workflow. When a workflow is called by a subflow activity, the workflow formal parameters permit the exchange of necessary data between calling and called process. A distinguishing characteristic of the WADE workflow engine is the **Delegation mechanism** that allows a set of agents to cooperatively execute a complex process. More in details the agent executing the calling workflow can decide to delegate the subflow to another agent on the basis of conditions evaluated at runtime. Such conditions may be related for instance to the current load (thus supporting the implementation of a GRID-like system) or to specific abilities required to carry out a portion of the whole process. The delegation mechanism is implemented by means of a an extension of the fipa-contract-net protocol (5) where the agent executing the calling workflow acts as initiator while the agent executing the subflow acts as responder. This protocol allows managing, when required, a process carried out by a set of cooperating agents as a single transaction.
- **Route Activity**: a route activity is an activity which performs no work processing, but simply supports routing decisions among its incoming and out coming transitions.

Finally, WADE introduces a few types of Activity, among them the **CodeActivity** and the **Web Services Activity**.

In a **Code Activity** the operations are specified directly by a piece of Java code embedded in the workflow process definition.

A Web Service Activity, instead, provides an easy way to invoke Web Services from a workflow. Two kinds of Web Services invocations are allowed: static and dynamic.

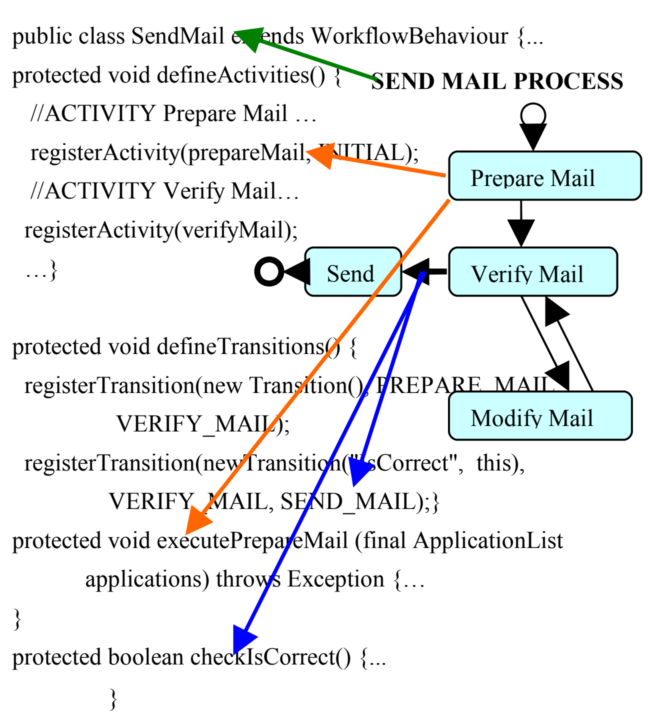
In the first case, it is necessary to import the WSDL describing the Web Service using WOLF. This operation generates a set of classes that will be used at workflow execution time to actually invoke the web service described by the imported WSDL. These classes can be extended by users with development skills, in order to satisfy some specific requirements (e.g.: logging some information before and after the invocation of the Web Service).

In the second case, the dynamic invocation is performed without the need of generating new classes. This approach is useful if the user doesn't need to customise these classes.

As mentioned a workflow is implemented by a Java class and it extends directly or indirectly the WADE class `WorkflowBehaviour`.

The `WorkflowBehaviour` class provides a set of APIs, consistent with the XPDL meta-model described in the previous section, which can be used by developers to implement their own workflows.

The mapping between the meta-model objects and the workflow implementation is shown in the example depicted in Figure 2.



```
public class SendMail extends WorkflowBehaviour {...
protected void defineActivities() {
  //ACTIVITY Prepare Mail …
  registerActivity(prepareMail, INITIAL);
  //ACTIVITY Verify Mail…
 registerActivity(verifyMail);
 …}

protected void defineTransitions() {
 registerTransition(new Transition(), PREPARE_MAIL,
          VERIFY_MAIL);
 registerTransition(new Transition("IsCorrect",  this),
          VERIFY_MAIL, SEND_MAIL);}
protected void executePrepareMail (final ApplicationList
          applications) throws Exception {…
}
protected boolean checkIsCorrect() {...
          }
```

Figure 2 – Mapping between meta – model objects and workflow implementation

The methods `registerActivity()` and `registerTransition()` must be used respectively to add an activity and a transition.

When the `registerActivity()` method is called, its first parameter is an instance of the activity to be added.
The actual tasks to be performed by an activity (no matter of its type) are specified in a `void` method of the workflow class; this method must have the same name of the activity, preceeded by the prefix "execute" (`execute<ActivityName>`). The workflow engine is in charge of invoking that method when the activity is visited.
In the same way, the `registerTransition()` method takes a transition instance as parameter. For transitions with conditions, the `boolean` expression to be evaluated by the workflow engine is specified in a boolean method that has same name of the condition, preceeded by the prefix "check" (`check<ConditionName>`).

An important feature of the object-oriented programming languages is the possibility to reuse the code by means of **inheritance mechanisms**. In order to bring the programming languages power also to the workflow representation, it has been choosen to provide the workflow with the inerithance mechanism too. Therefore it is possible to define a new workflow extending an old one, and then adding/removing activities and transitions. The overriding of methods associated to the activities and to the conditions of transitions is also permitted.

4. CONCLUSIONS

In this paper we described WADE, an open source framework to develop distributed applications based on the agent programming paradigm. WADE is based on JADE and in particular adds to it the possibility of modeling the agents' behaviours following the workflow metaphor. Moreover some administration and fault tolerance features are provided too.

As mentioned, in WADE a workflow is represented as a java class and WOLF provides a graphical view of it, making available to the developers both the expressiveness of a visual representation and the power of usual programming languages.

As well known, the workflow metaphor is traditionally used in the BPM context for web service orchestration. Because WADE provides support for Web Service invocation, it can be used even in this context, but its actual challenge is to bring the workflow approach from the business process level to the level of system internal logics. A direct consequence of the approach described is that the full power of WADE can be exploited for applications that imply the execution of possibly long and fairly complex tasks.

Furthermore, unlike the majority of existing workflow systems that provide a powerful centralized engine, in WADE each agent can embed a "micro workflow engine" and therefore a complex process can be carried out by a set of cooperating agents through the delegation mechanism.

From an industrial point of view WADE can be particularly useful to develop applications with strong requirements of both performance and scalability and high flexibility in defining the systems' logics.

5. REFERENCES

[1] JADE - Java Agent Development framework. http://jade.tilab.com

[2] FIPA – Foundation for Intelligent Physical Agents

http://www.fipa.org

[3] FIPA – The FIPA Contract Net interaction protocol

[4] XPDL XML Process Definition Language, http://www.wfmc.org/standards/xpdl.htm

[5] Shapiro, R. 2002. A comparison of XPDL, BPML and BPEL4WS (Rough Draft), Cape Vision

[6] BPMN Business Process Modeling Notation http://www.bpmn.org/

[7] G. Caire "WADE: An Open Source Platform for Workflows and Agents"

http://jade.tilab.com/wade/doc/tutorial-aamas2008.zip

[8] Caire G., Gotta D., Banzi "WADE: A software platform to develop mission critical applications exploiting agents and workflows", M., Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008) – Industry and Applications Track, Berger, Burg, Nishiyama (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 29-36.

[9] "WADE User Guide"

http://jade.tilab.com/wade/doc/WADE-User-Guide.pdf

[10] G. Caire, M. Porta, E. Quarantotto, G. Sacchi " Wolf – An Eclipse Plug-in for WADE"