Engineering JADE Agents with the Gaia Methodology

Pavlos Moraïtis [1], Eleftheria Petraki[2], Nikolaos I. Spanoudakis[3]

[1]Dept. of Computer Science
University of Cyprus
P.O. Box 20537, CY-1678 Nicosia, Cyprus
moraitis@ucy.ac.cy

[1,2,3]Singular Software,
26th October 43, 54626, Thessaloniki, Greece
{epetraki, nspan}@si.gr

Agent Oriented Software Engineering (AOSE) is one of the fields of the agent domain with a continuous growing interest. The reason is that the possibility to easily specify and implement agent-based systems is of a great importance for the recognition of the add-value of the agent technology in many application fields. In this paper we present an attempt towards this direction, by proposing a kind of roadmap of how one can combine the Gaia methodology for agent-oriented analysis and design and JADE, a FIPA compliant agent development framework, for an easer analysis, design and implementation of multi-agent systems. Our objective is realized through the presentation of the analysis, design and implementation phases, of a limited version of a system we currently develop in the context of the IST IMAGE project.

1. Introduction

During the last few years, there has been a growth of interest in the potential of agent technology in the context of software engineering. This has led to the proposal of several development environments to build agent systems (see for example Zeus [3], AgentBuilder [10], AgentTool [5], RETSINA [11], etc), software frameworks to develop agent applications in compliance with the FIPA specifications (see for example FIPA-OS [7], JADE [2], etc). These development environments and software frameworks demanded that system analysis and design methodologies, languages and procedures would support them. As a consequence, many of these were proposed along with a methodology (e.g. Zeus [4], AgentTool [12]) while in parallel have been proposed some promising agent-oriented software development methodologies, as Gaia [13], AUML [1], Tropos [8], MASE [12]. Also, the Aspect Oriented Programming [9] can be used as a methodology for design and implementation of agent role models. However, despite the possibilities provided by these methodologies, we believe that a further progress must be made, so that agent-based technologies realize their full potential, concerning the full covering of the software life cycle and the proposal of standards to support agent interoperability.

In this paper we present an attempt to use Gaia in order to engineer a multi-agent system (MAS) that is to be implemented with the JADE framework. The only

pretension we have with this paper is to share our experience to conceive and develop a MAS, by combining Gaia and JADE, in the context of the IST IMAGE project, with people who are interested in the development of real life agent-based systems. The Gaia methodology can be applied in a high level design. There is no given way to go from a Gaia model to a system design model. System implementation is still done through object-oriented techniques. Thus, the aim of this paper is to describe a kind of roadmap for implementing a Gaia model using the JADE framework. Towards this end, we provide some additional modeling techniques and make some slight modifications to the Gaia original specification, without obviously altering its philosophy and concepts.

This paper is organized in the following way. In sections 2 and 3 we briefly present the Gaia methodology and JADE framework. In section 4 we provide a sample Gaia model. In section 5 we provide a methodology for converting the Gaia model to a JADE implementation. Moreover, we propose some models useful for the detailed design phase. Finally, we discuss on AOSE.

2. Gaia Overview

The Gaia methodology is an attempt to define a complete and general methodology that it is specifically tailored to the analysis and design of MASs. Gaia is a general methodology that supports both the levels of the individual agent structure and the agent society in the MAS development process. MASs, according to Gaia, are viewed as being composed of a number of autonomous interactive agents that live in an organized society in which each agent plays one or more specific roles. Gaia defines the structure of a MAS in terms of a role model. The model identifies the roles that agents have to play within the MAS and the interaction protocols between the different roles.

The objective of the Gaia analysis process is the identification of the roles and the modeling of interactions between the roles found. Roles consist of four attributes: responsibilities, permissions, activities and protocols. Responsibilities are the key attribute related to a role since they determine the functionality. Responsibilities are of two types: liveness properties – the role has to add something good to the system, and safety properties – the role must prevent and disallow that something bad happens to the system. Liveness describes the tasks that an agent must fulfill given certain environmental conditions and safety ensures that an acceptable state of affairs is maintained during the execution cycle. In order to realize responsibilities, a role has a set of permissions. Permissions represent what the role is allowed to do and in particular, which information resources it is allowed to access. The activities are tasks that an agent performs without interacting with other agents. Finally, protocols are the specific patterns of interaction, e.g. a seller role can support different auction protocols. Gaia has formal operators and templates for representing roles and their attributes and also it has schemas that can be used for the representation of interactions between the various roles in a system.

The operators that can be used for liveness expressions-formulas along with their interpretations are presented in Table 1. Note that in liveness formulas activities are written underlined.

Table 1. Gaia Operators for Liveness Formulas

Operator	Interpretation
x . y	x followed by y
x | y	x or y occurs
x*	x occurs 0 or more times
x+	x occurs 1 or more times
x^{ω}	x occurs infinitely often
[x]	x is optional
x || y	x and y interleaved

In the Gaia design process the first step is to map roles into agent types and to create the right number of agent instances of each type. An agent type can be an aggregation of one or more agent roles. The second step is to determine the services model needed to fulfill a role in one or several agents. A service can be viewed as a function of the agent and can be derived from the list of protocols, activities, responsibilities and the liveness properties of a role. Finally, the last step is to create the acquaintance model for the representation of communication between the different agents. The acquaintance model does not define the actual messages that are exchanged between the agents it is rather a simple graph that represents the communication pathways between the different agent types.

3. JADE Overview

JADE is a software development framework fully implemented in JAVA language aiming at the development of multi-agent systems and applications that comply with FIPA standards for intelligent agents. JADE provides standard agent technologies and offers to the developer a number of features in order to simplify the development process:

- Distributed agent platform. The agent platform can be distributed on several hosts, each one of them executes one Java Virtual Machine.
- FIPA-Compliant agent platform, which includes the Agent Management System the Directory Facilitator and the Agent Communication Channel.
- Efficient transport of ACL messages between agents.

All agent communication is performed through message passing and the FIPA ACL is the language that is used to represent the messages. Each agent is equipped with an incoming message box and message polling can be blocking or non-blocking with an optional timeout. Moreover, JADE provides methods for message filtering. The developer can apply advanced filters on the various fields of the incoming message such as sender, performative or ontology.

FIPA specifies a set of standard interaction protocols such as FIPA-request, FIPA-query, etc. that can be used as standard templates to build agent conversations. For

every conversation among agents, JADE distinguishes the role of the agent that starts the conversation (initiator) and the role of the agent that engages in a conversation started by another agent (responder). According to the structure of these protocols, the initiator sends a message and the responder can subsequently reply by sending a not-understood or a refuse message indicating the inability to achieve the rational effect of the communicative act, or an agree message indicating the agreement to perform the communicative act. When the responder performs the action he must send an inform message. A failure message indicates that the action was not successful. JADE provides ready-made behaviour classes for both roles, following most of the FIPA specified interaction protocols. Because the FIPA interaction protocols share the same structure, JADE provides the AchieveREInitiator/Responder classes, a single homogeneous implementation of interaction protocols such as these mentioned above. Both classes provide methods for handling all possible protocol states.

In JADE, agent tasks or agent intentions are implemented through the use of behaviours. Behaviours are logical execution threads that can be composed in various ways to achieve complex execution patterns and can be initialized, suspended and spawned at any given time. The agent core keeps a task list that contains the active behaviours. JADE uses one thread per agent instead of one thread per behaviour to limit the number of threads running in the agent platform. A scheduler, hidden to the developer, carries out a round robin policy among all behaviours available in the queue. The behaviour can release the execution control with the use of blocking mechanisms, or it can permanently remove itself from the queue in run time. Each behaviour performs its designated operation be executing the core method action().

Behaviour is the root class of the behaviour hierarchy that defines several core methods and sets the basis for behaviour scheduling as it allows state transitions (starting, blocking and restarting). The children of this base class are SimpleBehaviour and CompositeBehaviour. The classes that descend from SimpleBehaviour represent atomic simple tasks that can be executed a number of times specified by the developer. Classes descending from CompositeBehaviour support the handling of multiple behaviours according to a policy. The actual agent tasks that are executed through this behaviour are not defined in the behaviour itself, but inside its children behaviours. The FSMBehaviour class, which executes its children behaviours according to a Finite State Machine (FSM) of behaviours, belongs in this branch of hierarchy. Each child represents the activity to be performed within a state of the FSM, with the transitions between the states defined by the developer. Because each state is itself a behaviour it is possible to embed state machines. The FSMBehaviour class has the responsibility of maintaining the transitions between states and selects the next state for execution. Some of the children of an FSMBehaviour can be registered as final states. The FSMBehaviour terminates after the completion of one of these children.

4. A Gaia model

In order to better understand our proposal on how GAIA and JADE can be combined to conceive and implement a multi-agent system (MAS) we will present a limited version of the system that is currently being implemented in the framework of the IST IMAGE project. We will show how this system can be analyzed, designed and implemented. For this system we have defined the following requirements:

- A user can request a route from one place to another. He can select among a variety of routes that are produced by the Geographical Information System (GIS).
- The MAS maintains a user profile so that it can filter the routes produced by the GIS and send to the user those that most suit him. The profiling will be based on criteria regarding the preferred transport type (private car, public transport, bicycle, on foot) and the preferred transport characteristics (shortest route, fastest route, cheapest route, etc).
- The system keeps track on selected user routes aiming:
  - To receive traffic events (closed roads) and check whether they affect the user's route (if that is the case then inform the user).
  - To adapt the service to user habits and needs.

  In the following sections this MAS will be analyzed, designed and implemented.

4.1. The Analysis phase

The analysis phase has led to the identification of four roles: one role, called EventsHandler, that handles traffic events, one role called TravelGuide that wraps the GIS, one role, called PersonalAssistant, that serves the user and, finally, a social type role, called SocialType, that should be taken by all agents. A Gaia roles model for our system is presented in Table 2.

Table 2. The Gaia Roles Model

Role: EventsHandler **Description**: It acts like a monitor. Whenever a new traffic event is detected it forwards it to all personal assistants. **Protocols and Activities**: CheckForNewEvents, InformForNewEvents. **Permissions**: read on-line traffic database, read acquaintances data structure. **Responsibilities**: **Liveness**: EVENTSHANDLER = (PushEvents)$^{\omega}$ PUSHEVENTS = CheckForNewEvents. InformForNewEvents **Safety**: A successful connection with the on-line traffic database is established.
Role: TravelGuide **Description**: It wraps a Geographical Information System (GIS). It can query the GIS for routes, from one point to another. **Protocols and Activities**: RegisterDF, QueryGIS, RequestRoutes, RespondRoutes. **Permissions**: read GIS. **Responsibilities**: **Liveness**: TRAVELGUIDE = RegisterDF. (FindRoutes)$^{\omega}$ FINDROUTES = RequestRoutes. QueryGIS. RespondRoutes **Safety**: A successful connection with the GIS is established.

Role: PersonalAssistant
Description: It acts on behalf of a profiled user. Whenever the user wants to go somewhere it gets the available routes and determines which routes best match the user's profile. These routes are presented to the user. Moreover, it can adapt (i.e. using learning capabilities) to a user's habits by learning from user selections. Finally, it receives information on traffic events, it checks whether such events affect its user's route and in such a case it informs the user.
Protocols and Activities: InitUserProfile, UserRequest, InferUserNeeds, PresentRoutes, LearnByUserSelection, CheckApplicability, PresentEvent, RequestRoutes, RespondRoutes, InformForNewEvents.
Permissions: create, read, update user profile data structure, read acquaintances data structure.
Responsibilities:
Liveness:
PERSONALASSISTANT = InitUserProfile. ((ServeUser) $^{\omega}$ || (ReceiveNewEvents) $^{\omega}$)
RECEIVENEWEVENTS = InformForNewEvents. CheckApplicability. [PresentEvent]
SERVEUSER = UserRequest. RequestRoutes. RespondRoutes. [InferUserNeeds]. PresentRoutes. LearnByUserSelection
Safety: true

Role: SocialType
Description: It requests agents that perform specific services from the DF. It also gets acquainted with specific agents.
Protocols and Activities: RegisterDF, QueryDF, SaveNewAcquaintance, IntroductsNewAgents.
Permissions: create, read, update acquaintances data structure.
Responsibilities:
Liveness:
SOCIALTYPE = GetAcquainted. (MeetSomeone) $^{\omega}$
GETACQUAINTED = RegisterDF. QueryDF. [IntroductNewAgent]
MEETSOMEONE = IntroductNewAgent. SaveNewAcquaintance
Safety: true

Here it must be noted that another role is involved in the MAS operation. It is the Directory Facilitator (DF) FIPA role that is supported by JADE. However, this role concerns the operational level of the MAS and not the application itself, that's why a Gaia representation is not supplied for this role. Moreover, interactions with it are not presented as protocols, as they are defined in Gaia methodology, but as activities. Indeed, the activities RegisterDF (denoting the registration to the DF) and QueryDF (querying for agents of specific types or that have registered specific services activities) are DF services provided directly by JADE framework, provided not as a result of interaction between agents, but as method invocations.

The Gaia interaction model denotes which action returns from a request along with the roles that can initiate a request and the corresponding responders. Table 3 holds the necessary information for our model.

Table 3. Gaia Interactions Model

Protocol	IntroductNewAgent	InformForNewEvents	RequestRoutes
Initiator(s)	SocialType	EventsHandler	PersonalAssistant
Receiver(s)	SocialType	PersonalAssistant	TravelGuide
Responding Action	-	-	RespondRoutes
Purpose/ Parameters	Introduce an agent to other agents. Possible content is the services and name associated with the initiator agent.	Inform an assistant that a new traffic event has occurred.	The assistant agent requests a set of routes from one place to another. The response includes different routes with different characteristics (shortest, fastest, cheapest) and for different transportation (private car, public transport, on foot).

4.2. The Design phase

During this phase the Agent model is achieved, along with the services and acquaintance models.

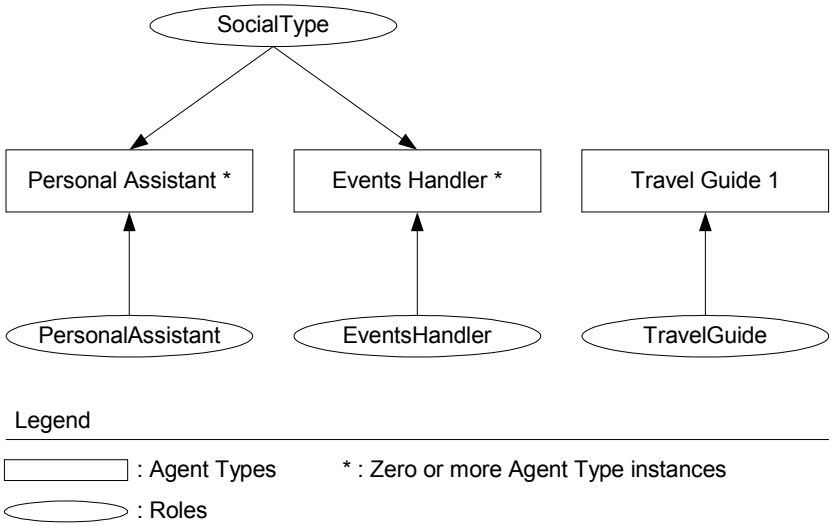

Fig. 1. Gaia Agent Model

The Agent model creates agent types by aggregating roles. Each emerging agent type can be represented as a role that combines all the aggregated roles attributes (activities, protocols, responsibilities and permissions). The agents model for our system will include three agent types: the personal assistant agent type, who fulfills the PersonalAssistant and SocialType roles, the events handler agent type, who fulfills the EventsHandler and SocialType roles and the travel guide agent type, who fulfills the TravelGuide role.
There will be one travel guide agent, as many personal assistants as the users of the system and zero or more events handlers. The Agent model is presented graphically in Figure 1.

The services model for our system is presented in Table 4.

Table 4. Gaia Services Model

Service	Obtain route	Get notified on a relevant to the user's route traffic event
Inputs	Origin, destination	-
Outputs	A set of routes	The description of the event
Pre-condition	A personalized assistant agent is instantiated and associated with the user	A personalized assistant agent is instantiated and associated with the user. The user has selected a route to somewhere. A traffic event that is relevant to the user's route has happened
Post-condition	User selects a route	-

Finally we define the acquaintances model. For this model we propose a slight modification compared to the original definition presented in Gaia.

Table 5. Gaia Acquaintances Model

	Personal Assistant	Travel Guide	Events Handler
Personal Assistant		I, A	I
Travel Guide	I		
Events Handler	I, A		

Legend:
I: Interacts (read "I" occurrences in rows, e.g. the personal assistant agent type interacts with travel guide and events handler agent types).
A: Is acquainted, has the agent type in his acquaintances data structure (read "A" occurrences in rows, e.g. the personal assistant agent type knows travel guide agent types).

We believe that this modification takes better into account the idea that an agent can interact with another agent (e.g. just responding to a request) without having necessarily any knowledge (information) about him. Therefore an analyst needs not only to specify which agent interacts with which, but which agent is acquainted with whom (i.e. knows whom) also. So, the personal assistants are acquainted and interact with travel guides and just interact with events handlers. Events handlers are acquainted and interact with personal assistants, while travel guides are not aware of the other agents however they interact with (service requests of) personal assistants. The above scheme is illustrated in Table 5.

At this point the abstract design of the system is complete, since the limit of Gaia has been reached. More effort must be done in order to obtain a good design though. At the end of the design process the system must be ready for implementation.

5. Developing JADE Agents from a Gaia Model

When moving from the Gaia model to an implementation using the JADE framework we have to make some assumptions and definitions.
Let's consider the liveness part of each role as its behaviour (usually having the same name with the role) in correspondence with the JADE terminology. Thus a simple or a complex behaviour represents each role. This behaviour is considered as the top-level behaviour of the role. Each behaviour may contain other behaviours, as in the JADE behaviours model. Let the contained behaviours be called lower level behaviours. The SocialType role of our system, for instance, has the SocialType top behaviour. This behaviour has two lower level behaviours, GetAcquainted and MeetSomeone.

The ω and || operators on Gaia liveness formulas now have the following meaning. The ω means that a lower level behaviour is added by the behavior that contains it in the Gaia liveness formula and is only removed from the agent's scheduler when the behavior that added it, is removed itself. If such behaviours are more than one, they are connected with the || symbol which denotes that they execute "concurrently". Concurrency in JADE agent behaviours is simulated. As noted before, only one thread executes per agent and behaviour actions are scheduled in a round robin policy.

5.1. Detailed Design

Many important design issues have yet to be covered when trying to implement a Gaia model with the JADE framework. Some of them are: a) ACL Messages (ontologies, protocols, content), b) Data structures, c) Algorithms and software components.

The ACL messages should be defined with respect to the FIPA ACL Message Structure Specification [6] and the JADE ACL Message class fields. The ACL Messages RequestRoutes and RespondRoutes are presented in Figure 2. It is obvious that FAILURE and REFUSE ACL messages should be defined for global use in cases that, an action failed, is not supported or is denied.

ACL Message: RequestRoutes	**ACL Message**: RespondRoutes
Sender: Personal Assistant Agent	**Sender**: Travel Guide Agent
Receiver: Travel Guide Agent	**Receiver**: Personal Assistant Agent
FIPA performative: REQUEST	**FIPA performative**: INFORM
Protocol: RequestRoutes	**Protocol**: RequestRoutes
Language: SL	**Language**: SL
Ontology: ImageOntology	**Ontology**: ImageOntology
Content: Ontology action: RequestRoutes	**Content**: Ontology concept: Routes

Fig. 2. ACL Messages Definition

Finally, the designer of the MAS will easily implement agents whose internal structures and methods are pre-defined. To that end, the data structures and the AI tools that are to be used should be defined in this stage. For our system the following structures and methods should be clarified at this point:

- The acquaintances structure: It will contain information about other agents. Some of this information might include names, types (or services) and addresses (as will see in the following in JADE address and name are quite the same). The SocialType role maintains this structure (see the permissions field of the role definition in Table 2).
- The user profile structure: What information will be known about the user, how is it organized. Such questions must be answered at this point. The PersonalAssistant role maintains this structure (see the permissions field of the role definition in Table 2).
- The route structure: What is a route, what attributes are associated with a route. This structure is needed by both the TravelGuide and PersonalAssistant roles, the former instantiates such objects by information that it gets from the GIS (QueryGIS activity), while the latter filters the route structure objects according to the user profile (InferUserNeeds activity).
- The traffic event structure: What is a traffic event, what attributes are associated with it, how is it associated with a route. Both the EventsHandler and PersonalAssistant roles use this structure. The former instantiates such objects by information that it gets from external sources (CheckForNewEvents activity), while the latter checks whether a traffic event structure object is in a user's active route (CheckApplicability activity).
- The learning method: What will be learned about the user, how, where is it going to be stored, which machine learning algorithm will be used (different goals can indicate different algorithms). A learning method will be used by the LearnByUserSelection activity of the PersonalAssistant role.
- The components and technologies that will enable communication with external systems. Such systems are the on-line traffic database and the GIS. If the GIS services are available as web services then a suitable SOAP (Simple Object Access

Protocol) client must be developed along with an XML parser that will translate the SOAP message content to an ontology concept or a JAVA object.

5.2. The JADE Implementation

At this point the MAS designer should have the full plan on how to implement the system. In our case the framework is JADE and the purpose of this paragraph is precisely to explain how the Gaia model is translated to a JADE implementation.

The procedure is quite straightforward. All Gaia liveness formulas are translated to JADE behaviours. Activities and protocols can be translated to JADE behaviours, to action methods (which will be part of finite state machine - FSM like behaviours) or to simple methods of behaviours. The JADE behaviours that can be useful for our model are the SimpleBehaviour, FSMBehaviour, AcheiveREResponder and AchieveREInitiator.

The behaviours that start their execution when a message arrives, can receive this message either at the beginning of the action method (simple behaviours) or by spawning an additional behaviour whose purpose is the continuous polling of the message box (complex behaviours). For behaviours that start by a message from the Graphical User Interface (GUI), a GUI event receiver method should be implemented on the agent that starts the corresponding behaviour. Finally, those behaviours that start by querying a data source, or by a calculation, should be explicitly added by their upper level behaviour. For example, the SocialType role adds both the GetAcquained and the MeetSomeone behaviours. The difference is that GetAcquained will set itself as finished after executing once, while the MeetSomeone will continue executing forever - or until the agent is "killed".

The safety properties of the Gaia roles model must be taken into account when designing the JADE behaviours. Some behaviours of the role, in order to execute properly, require the safety conditions to be true. Towards that end, one at least behaviour is responsible for monitoring each safety condition of a role. Whenever a safety condition is found to be false, the functionality of the behaviours that depend on this safety condition is suspended and the monitoring behaviour initializes a procedure that will reestablish the validity of safety conditions. This procedure, for instance, can be the addition to the agent scheduler of a specific behaviour that will address the task of restoring the validity of safety conditions. In general, this procedure depends on the nature of the implemented system and the safety conditions. When the safety conditions are restored, the suspended functionalities are reactivated.

In our case, the safety requirement of the TravelGuide role is the establishment of communication with the GIS. The FindRoutes behaviour is responsible for monitoring the validity of this safety requirement. Whenever a connection fails to be established the FindRoutes behaviour sends to the agent GUI an event that results in a connection failure message, while responding to the personal assistant agent with a FAILURE ACL message. The system administrator must act in order to restore the GIS communication.

All behaviours of the lowest level are implemented first:

- PushEvents: A SimpleBehaviour that queries a database and if it gets a new event prepares an ACL message and sends it to all personal assistant agents.

- FindRoutes: It is a SimpleBehaviour that waits until it receives a specific ACL message, queries the GIS and sends back to the original sender a responding ACL message.
- ReceiveNewEvents: A SimpleBehaviour that waits until it receives a specific ACL message verifies if it is of interest for the specific user and sends an appropriate event to the GUI.
- ServeUser: a complex behaviour more like an FSMBehaviour with three states. At the first state it gets the user request (it is added to the agent scheduler as a consequence of that request) and sends an ACL message to the travel guide agent. Then it waits for its response. Alternatively, after getting the user request it could add an AchieveREInitiator behaviour. When it gets the response (second state) it infers on which routes should be forwarded to the user, forwards them and terminates its execution. If the user selects a route through the GUI, the GUI event catcher method of the agent starts this behaviour, but sets it immediately at the third state, which employs the learning algorithm in order to gain knowledge from the user action.
- GetAcquainted: This is a SimpleBehaviour that registers the agent to the DF, gets all needed agents from the DF and finally sends appropriate IntroductNewAgent ACL messages to all agents whom this agent wants to notify about his appearance. After the execution of these tasks the behaviour removes itself.
- MeetSomeone: a SimpleBehaviour that waits until it receives a specific ACL message then updates its acquaintance data structure with a new contact and the services that the new contact provides.

A good architecture paradigm contains no functionality at the top-level behaviour, instead, the agents tasks are embedded in lower level behaviors. Thus, the top-level behaviours that represent the actions performed in the setup phase of the agent are:

- EventsHandlerAgent: initialize the Acquaintances data structure, add the PushEvents, GetAcquainted and MeetSomeone behaviours.
- TravelGuideAgent: register to the DF and add the FindRoutes behaviour.
- PersonalAssistantAgent: initialize the Acquaintances data structure, get the initial user profile, add the GetAcquainted, MeetSomeone, ServeUser and ReceiveNewEvents behaviours.

Summarizing, the following steps should be followed in order to easily translate a Gaia model to a JADE implementation:

1. Define all the ACL messages by using the Gaia protocols and interactions models.
2. Design the needed data structures and software modules that are going to be used by the agents by using the Gaia roles and agents models.
3. Decide on the implementation of the safety conditions of each role.
4. Define the JADE behaviours. Start by implementing those of the lowest levels, using the various Behaviour class antecedents provided by JADE. The Gaia model that is useful in this phase is the roles model. Behaviours that are activated on the receipt of a specific message type must either add a receiver behaviour, or receive a message (with the appropriate message filtering template) at the start of their action. Gaia activities that execute one after another (sequence of actions that require no interaction between agents) with no interleaving protocols can be aggregated in one activity (behaviour method or action). However, for reusability, clarity and programming tasks allocation reasons, we believe that a developer

could opt to implement them as separate methods (or actions in an FSM like behaviour).
5. Keep in mind that Gaia roles translated to JADE behaviours are reusable pieces of code. In our system, the same code of the behaviours `GetAcquainted` and `MeetSomeone` will be used both for the `personal assistant` and `events handler` agents.
6. At the setup method of the Agent class invoke all methods (Gaia activities) that are executed once at the beginning of the top behaviour (e.g. `RegisterDF`). Initialize all agent data structures. Add all behaviours of the lower level in the agent scheduler.

6. Discussion

In this paper we have presented the analysis, design and implementation phases of a limited version of a system developed in the context of the IST IMAGE project. As we already have said before, the only pretension we have with this paper is to share our experience on how one can combine the Gaia methodology and the JADE development environment in order to implement a real multi-agent system. Gaia methodology is an easy to use agent-orient software development methodology that however presently, covers only the phases of analysis and design. On the other hand JADE is a FIPA specifications compliant agent development environment that gives several facilities for an easy and fast implementation. Our aim was to reveal the mapping that may exists between the basic concepts proposed by Gaia for agents specification and agents interactions and those provided by JADE for agents implementation, and therefore to propose a kind of roadmap for agents developers. Presently we have introduced a slight modification for the Gaia acquaintances model and our future work, through our main work on IMAGE project, will be to examine if there could be proposed some modifications in both, Gaia and JADE, that would help to make more efficient their combination.

Acknowledgements

We gratefully acknowledge the Information Society Technologies (IST) Programme and specifically the Research and Technological Development (RTD) “Intelligent Mobility Agent for Complex Geographic Environments” (IMAGE, IST-2000-30047) project for contributing in the funding of our work.

References

1. Agent UML: http://www.auml.org/
2. Bellifemine, F., Caire, G., Trucco, T., Rimassa, G.: Jade Programmer’s Guide. JADE 2.5 (2002) http://sharon.cselt.it/projects/jade/
3. Collis, J. and Ndumu, D.: Zeus Technical Manual. Intelligent Systems Research Group, BT Labs. British Telecommunications. (1999)

4. Collis, J. and Ndumu, D.: Zeus Methodology Documentation Part I: The Role Modelling Guide. Intelligent Systems Research Group, BT labs. British Telecommunications (1999)
5. DeLoach S. and Wood, M.: Developing Multiagent Systems with agentTool. In: Castelfranchi, C., Lesperance Y. (Eds.): Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop (ATAL 2000, Boston, MA, USA, July 7-9, 2000),. Lecture Notes in Computer Science. Vol. 1986, Springer Verlag, Berlin (2001)
6. FIPA specification XC00061E: FIPA ACL Message Structure Specification (2000) http://www.fipa.org
7. FIPA-OS: A component-based toolkit enabling rapid development of FIPA compliant agents: http://fipa-os.sourceforge.net/
8. Giunchiglia, F., Mylopoulos, J., Perini, A.: The Tropos Software Development Methodology: Processes, Models and Diagrams, in AAMAS02
9. Kendall, E.A.: Role Model Designs and Implementations with Aspect Oriented Programming. Proceedings of the 1999 Conference on Object- Oriented Programming Systems, Languages, and Applications (OOPSLA'99)
10. Reticular Systems Inc: AgentBuilder An Integrated Toolkit for Constructing Intelligent Software Agents. Revision 1.3. (1999) http://www.agentbuilder.com
11. Sycara, K., Paolucci, M., van Velsen, M. and Giampapa, J.: The RETSINA MAS Infrastructure. Accepted by the Journal of Autonomous Agents and Multi-agent Systems (JAAMS)
12. Wood, M.F. and DeLoach, S.A.: An Overview of the Multiagent Systems Engineering Methodology. AOSE-2000, The First International Workshop on Agent-Oriented Software Engineering. Limerick, Ireland (2000)
13. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems Vol. 3. No. 3 (2000) 285-312