A Conceptual and Computational Model for Knowledge-based Agents in ANDROID

Fabio Sartori, Lorenza Manenti and Luca Grazioli
Department of Informatics, Systems and Communication,
University of Milano-Bicocca,
Viale Sarca 336/14, 20126, Milano, Italy
Email: {sartori, manenti}@disco.unimib.it,
l.grazioli3@campus.unimib.it

***Abstract*—Today ANDROID is the most popular mobile operating system in the world: the development of ANDROID, together with the performance improvement offered by modern PDAs, like smart-phones and tablets, has allowed many users to know new kinds of applications that were not accessible to them in the recent past.**

In this paper we present a framework for programming agents in the ANDROID world, based on the Knowledge Artifact notion to develop knowledge-based systems.

This framework has been modeled as a client–server architecture, with the aim to show how the implementation of agents modeled on the basis of Knowledge Artifacts can help everyone to design, implement and use decision support systems for a specific domain, with many potential benefits in their day-by-day activities.

The framework application will be presented in a prototype to support operators of Italian Fire Corps and Civil Protection Department in critical situations, like geographically distributed fires and earthquakes management.

I. Introduction

Nowadays ANDROID is the most popular mobile operating system in the world, and reaches out to touch peaks of diffusion, in some countries, more than 90% of the total smartphone market[1]. The development of ANDROID together with the performance improvement offered by modern PDAs, like smartphones and tablets, has allowed many users to keep in touch with new kinds of applications that were not accessible to them in the recent past.

Among them, applications related to the Agent Oriented Programming (AOP) paradigm [1] are particularly influenced by the wide diffusion of personal devices, thanks to their intrinsic mobile nature. Different open-source frameworks devoted to the development of agent-based programs, like JADE[2] [2], JASON[3] [3] and CArtAgO[4] [4] have been recently imported into ANDROID by means of the implementation of specific add-ons or ad-hoc frameworks (e.g. [5], [6]).

In this paper we present a framework for programming agents in the ANDROID world, making it transparent to the programmer, basing it on the Knowledge Artifact [7] notion. While the *artifact* concept is often described as *the result of a disciplined human activity, following rules based on training and experience*, Knowledge Artifacts [11] are artifact specializations devoted to represent expert knowledge in object–manufacturing or service–delivery fields. The final aim of our Knowledge Artifact is generating executable rule–based systems written in JESS[5], minimizing the knowledge engineering effort. To this scope, correlated tools for the representation of functional and structural knowledge (i.e. ontologies [8]), procedural knowledge (i.e. influence nets [9]) and experiential knowledge (i.e. task structures [10]) have been integrated into a unique conceptual and computational framework, providing the user with an opportune set of primitives for designing and implementing decision support systems without a deep knowledge on the specific language syntax.

This framework has been modeled as a client–server architecture, where the server is a knowledge-based agent having two tasks, the creation of the rule-based system according to the user (i.e. the expert) indications and the execution of it according to the data sent it by the client, that is a sort of reactive agent sending inputs and receiving outputs from the knowledge-based agent: in this way, it was possible to overcome the impossibility, at the current state of JESS implementation[6] to import JESS library under the ANDROID OS.

The main aim of this paper is to show how the implementation of agents modeled on the basis of Knowledge Artifacts can help everyone to design, implement and use decision support systems for a specific domain, with many potential benefits in their day-by-day activities.

The most interesting feature of the framework is its capability to act as a CAKE (Computer-Aided Knowledge Engineering) environment: the implementation of KBSs has been always conceived as a very specific task, which can be only conducted by knowledge engineers with the support of domain experts. Knowledge engineering methodologies, such as CommonKads [12] and MIKE [13], have been proposed in the past as standard and generalized solutions to overcome

[1]Data extracted from http://www.androidworld.it/2013/04/29/android-in-italia-al-625-e-nel-mondo-al-642-secondo-le-ultime-stime-153115/

[2]Available at http://jade.tilab.com/

[3]Available at http://jason.sourceforge.net/wp/

[4]Available at http://cartago.sourceforge.net/

[5]Acronym of Java Expert System Shell, available at http://herzberg.ca.sandia.gov

[6]See the discussion at http://jess.2305737.n4.nabble.com/JESS-Re-Jess-jar-on-Android-td3957868.html

the knowledge acquisition and representation bottlenecks, but addressed to users highly skilled in the design of complex software systems.

The rest of the paper is organized as follows: Section II will introduce the agent-based framework and the conceptual model of Knowledge Artifact it is based on, also discussing the computational model of the agents developed according to it. Section III will present a case study to show how it can be profitably used in a specific domain. In the end, in Section IV a discussion on how this model can be implemented considering the AOP paradigm will be presented along with a discussion on the future developments and works.

II. The Agent-Based Framework and the KA Notion

In this Section we first present the agent–based framework describing the main components of the model; then, we discuss the conceptual model and the relative implementation of the KA notion according to the agent–based model has been developed.

A. Agent-based Framework

In relationship with the client-server architecture, our framework is composed of two main elements:

- simple reactive agents $a_1, .., a_i, .., a_n$ located on mobile devices and that perceived and collected information on the "state of the world". These information can be detected by means of device sensors (e.g. GPS, barometer, pressure altimeter and so on) and directly provided by the user of the system by means of an appropriate graphic user interface (GUI);
- a knowledge-based agent *KA-Agent* that is responsible for the management of its internal knowledge and the elaboration of information received from agents a_i. In particular the goal of a KA-Agent is twofold: (i) it has to interact with the domain expert by means of a graphic user interface (GUI), obtaining information about the domain in terms of concepts, relationships and rules, and creating the corresponding ontology, influence network and rule-based system; (ii) it has to interact with agents a_i and it has to elaborate the information they provide with the expert system previously created.

Fig. 1 graphically represents the components and the interactions in our framework: expert users interact with the KA-Agent GUI in order to create the expert system, while non-expert users interact with agent a_i GUI in order to provide information that will be used by KA-Agent in elaborating knowledge.

Since the KA-Agent represents the main relevant part of our system, in the follow we will focus on the KA notion upon which it is based on, considering the conceptual model and the relative implementation.

B. The Conceptual Model of Knowledge Artifact

In our approach, the Knowledge Artifact is described as a 3–tuple $\langle O, IN, TS \rangle$, where O is an ontology of the investigated domain, IN is an Influence Net to represent the causal dependencies among the ontology elements and TS are task structures to represent how one or more outputs can be produced by the system according to a rule–base system strategy.

In the KA model, the underlying ontology is a taxonomy: the root is the description of the problem to be solved, the inner nodes are system inputs or partial outputs and the leaves of the hierarchy are effective outputs of the system.

The Influence Net model is a structured process that allows to analyse complex problems of cause-effect type in order to determine an optimal strategy for the execution of certain actions, to obtain an optimal result. The Influence Net is a graphical model that describes the events and their causal relationships. Using information based on facts and experience of the expert, it is possible to analyze the uncertainties created by the environment in which we operate. This analysis helps the developer to identify the events and relationships that can improve or worsen the desired result. In this way you can determine the best strategy.

The Influence Net can be defined as a 4–tuple $\langle I, P, O, A \rangle$, where:

- I is the set of *input nodes*, i.e. the information needed to the KBS to work properly;
- P is the set or *partial output nodes*, i.e. the collection of new pieces of knowledge and information elaborated by the system to reach the desired output;
- O is the set of *output nodes*, i.e. the effective answers of the system to the described problem; outputs are values that can be returned to the user;
- A is the set of *arcs* among the nodes: an arc between two nodes specifies that a causal relationship exists between them; an arc can go from an input to a partial node or an output, as well as from partial node to another one or an output. Moreover, an arc can go from an output to another output. Every other kind of arcs is not permitted.

Finally, Task Structures allow to describe in a rule–based system way how the causal process defined by a given IN can be modeled. Each task is devoted to define computationally a portion of an Influence Net: in particular, *sub–tasks* are procedures to specify how a partial output is obtained, while *tasks* are used to explain how an output can be derived from one or more influencing partial outputs and inputs. A task cannot be completed until all the sub–tasks influencing it have been finished. In this way, the TS modeling allows to clearly identify all the levels of the system. The task and sub–task bodies are a sequence of rules, i.e. $LHS(LeftHandSide) -> RHS(RightHandSide)$ constructs.

Each LHS contains the conditions that must be verified so that the rule can be applied: it is a logic clause, which turns out to be a sufficient condition for the execution of the action indicated in the RHS. Each RHS contains the description of the actions to conduct as a result of the rule execution. The last step of our model is then the translation of all the task and sub–task bodies into production rules of a specific language (JESS in our case).

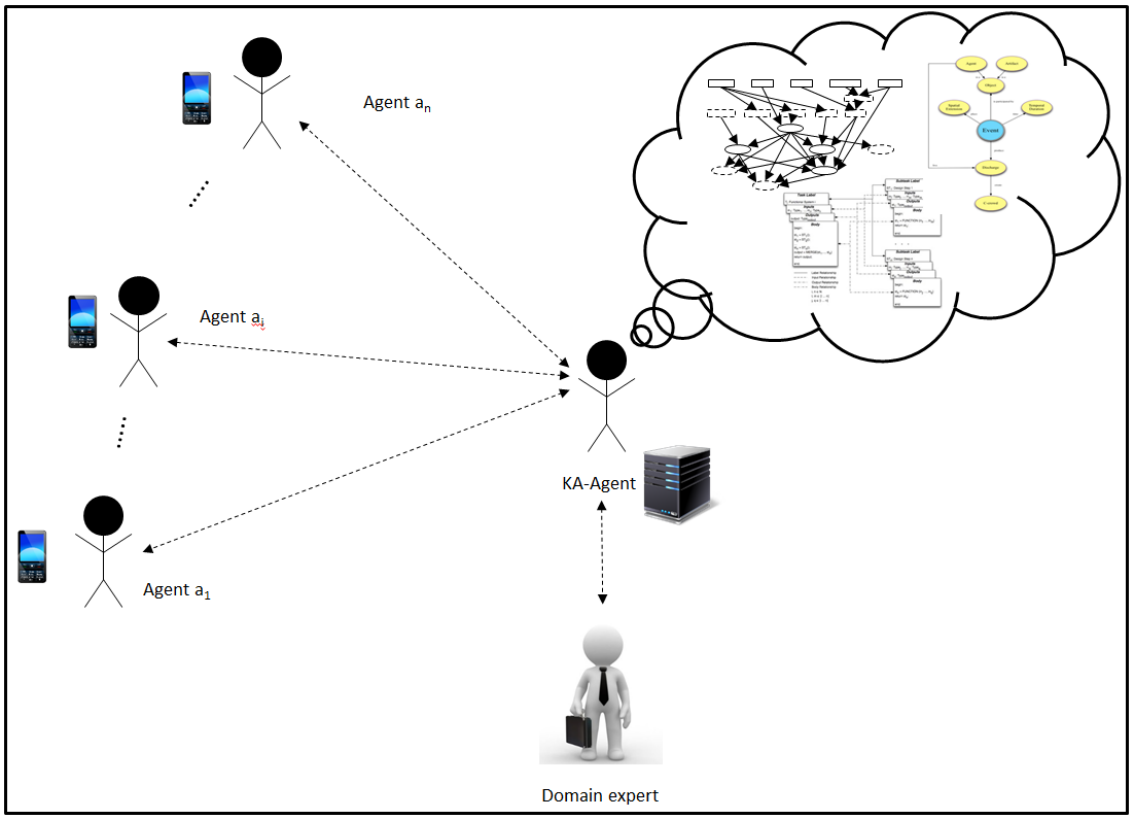

Fig. 1: Graphical representation of the agent-based model: agents a_i, KA-Agent, and interactions among them and with domain expert

C. Knowledge Artifact Implementation: the KA-Agent

The implementation of the different elements composing the knowledge engineering framework has exploited the XML language. A proper schema has been developed for each of them, as well as dedicated parsers to allow the user to interact with them. Following the conceptual model briefly introduced in the previous section, the first schema is the ontological one, defining opportune tags to specify *inputs*, where the name of the input can be put together with a description and a value for it. Moreover, it is possible to define an $\langle affects \rangle$ relationship for each input, in order to explain how it is involved in the next steps of the elaboration (i.e. which output or partial output does it contribute to state?).

A *partial output* (i.e. an inner node between an input and a leaf of the taxonomy) is limited by a specific pair of tags. The fields are the same as the input case, with the difference that a partial output can be influenced by another entity too: this is the sense of the $\langle influencedBy \rangle$ relationship. Finally, the $\langle output \rangle$ tag allows to describe completely an effective output of the system, i.e. a leaf of the taxonomy developed to represent the problem domain.

To produce an Influence Net, the taxonomy is bottom–up parsed, in order to identify the right flow from inputs to outputs by navigating the *influencedBy* relationships designed by the user. In this way, different portions of the under development system can be described. Outputs, partial outputs and inputs are bounded by arcs which specify the *source* and the *target* nodes.

Finally, an XML schema for the *task* (*subtask*) element of the framework have been developed. The parser produces an XML file for each output considered in the Influence Net. The tags *input* and *subtask* allows to define which inputs and partial outputs are needed to the output represented by the task to be produced. The *body* tag is adopted to model the sequence of rules necessary to process inputs and results returned by influencing subtasks: a rule is composed of an $\langle if \rangle$ ... $\langle do \rangle$ construct, where the if statement permits to represent the LHS part of the rule, while the do statement concerns the RHS part of the rule.

The XML files introduced so far have been incorporated into the KA–Agent knowledge base: they allow the definition of a rule–based system to solve problems in specific domains. The developed GUI permits the user to interact with the KA–Agent to design the underlying taxonomy, influence net and tasks/subtasks. Moreover, it is possible to transform the task into a collection of files containing rules written in the JESS language: Figure 2 shows a sketch of the supporting tool for this scope.

III. A Case Study

Once the KA–Agent has been created and programmed as a rule–based system, it is able to interact with one or more

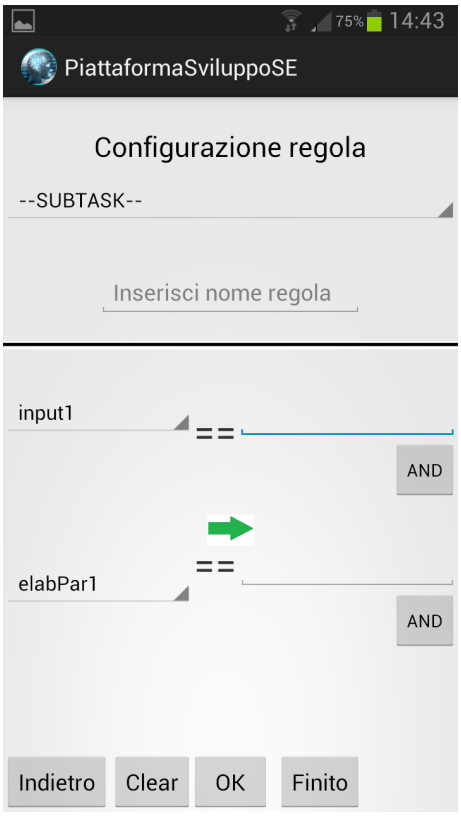

Fig. 2: The KA–Agent GUI for supporting domain experts in the creation of rule–based system from task structures

clients to receive inputs and send outputs. Clients are reactive agents in our model: they are spatially distributed and detect observations about the knowledge domain under investigation. They can send to the KA–Agent these observations in order to get suggestions about the action to take. These suggestions are elaborated by the KA–Agent according to its KA model. This scenario has been successfully tested in a case study inspired us by the *STOP*[7] handbook supplied to the Italian Fire Corps and Civil Protection Department of the Presidency of Council of Ministers for the construction of safety building measures for some building structures that have been damaged by an earthquake. After L'Aquila's earthquake in 2009, this document has been prepared in order to standardize the steps to follow in similar situations.

The structure of this document is suitable for the creation of a rule-based system, being modular and with specific and well defined cases. Using the created application, two knowledge models have been created (the first for the safety of the walls through rakers, the other for the safety of the openings through special scaffoldings).

A. The STOP–Agent: an ANDROID Client

Every operator involved in the emergency procedures to make safe buildings and infrastructures is provided with an ANDROID application on his/her smartphone: this application has been modeled as a reactive agent, namely the *STOP–Agent*, communicating with the KA–agent via the client–server architecture introduced above: Figure 3 shows the interface for its initialization. Each STOP–Agent sends to the KA–agent data about the conditions of the site it is analyzing: according to the STOP handbook, these data allow to make considerations about the real conditions of the building walls after the earthquake in order to understand which raker or scaffolding to adopt.

Exploiting the ANDROID primitives, it has been possible to create a stable mechanism for the communication with the server. In particular, the following tools were useful to implement the STOP–Agent:

- *activities*: a class that extends an *Activity* class is responsible for the communication with the user, to support him/her in setting the layout, assigning the listeners to the various widgets (ANDROID's graphical tools) and setting the context menu;
- *listener*: a class that implements the interface *OnClickListener* is a listener. An instance of this object is always associated with a widget;
- *asyncTask*: a class that extends *AsyncTask* is an asynchronous task that performs some operations concurrently with the execution of the user interface (for example the connection to a server must be carried out in a AsyncTask

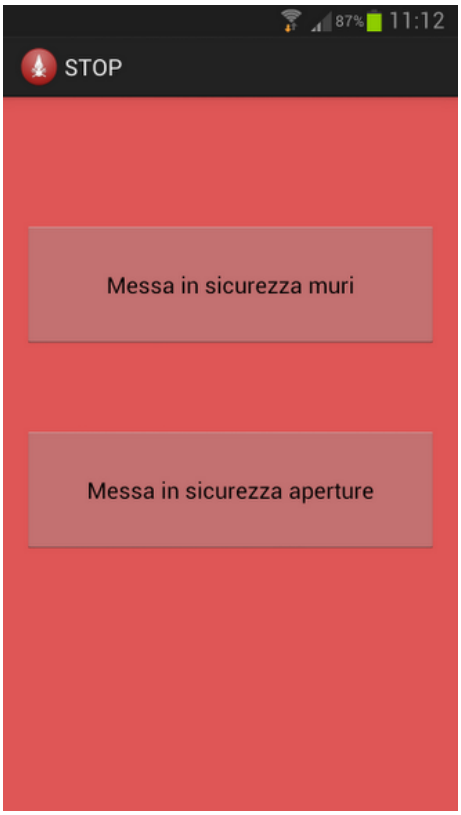

Fig. 3: The STOP–Agent GUI for the activation of the two application knowledge models

[7]Acronym of *Schede Tecniche di Opere Provvisionali*, see http://www.vigilfuoco.it/aspx/notizia.aspx?codnews=8293 for details.

instance, not in an Activity one);

The typical mechanism to interface the client and the server is the following one: the Activity object prepares the layout and sets the widgets' listeners and a container with the information useful for the server; then, it possibly starts the AsyncTask instance for sending the correct request to the server, passing to it the previously created container. Before starting the asynchronous task, in most cases, the listener activates a dialog window that locks the user interface in waiting for the communication with the server; the AsyncTask predisposes the necessary Sockets for the communication and then performs its request to the server, sending the information about the case study observation (see Figure 4) enclosed in the container. Before concluding, it closes (dismisses) the waiting dialog window.

B. The KA–Agent: a JAVA Server

The KA–Agent creates an instance of the KA model for each active STOP–Agents in communication with it. Then, it executes the model according to the rule–based system previously generated and sends answers to the STOP–Agent that will be able to take the proper action, as shown in Figure 5.

The design of the server exploits an existent framework for the rule-based systems creation. This framework allows to produce a *.clp* file runnable under JESS and that contains the rules describing the behavior of the rule-based system. A *Controller* class has been added to this framework to build up the interface between all the server's classes and the preexistent knowledge engineering environment.

The server, once activated, can accept both requests for the creation of a new system and for the resolution of problems on the basis of existing rule-based systems. To do this, concurrent programming is used: the server manages the different requests concurrently, through an opportune thread, namely *MonitorThread*. A MonitorThread instance starts the thread in charge of listening the requests for the creation of a new rule-based system (i.e. *GestoreThread*). Moreover, MonitorThread allows to properly manage the ports on which other threads will interface, and provides methods necessary for their correct startup. Another thread, namely *ManagingThread* is instanced by GestoreThread for the use of previously created rule–based systems. This thread manages the .clp files archive, being sure that inputs and outputs are correctly received and sent by the right system and JESS' libraries are correctly invoked.

IV. Conclusion and Future Works

This paper has presented an ongoing research project aiming to design and implement tools for supporting the user in the development of knowledge-based systems. This framework is based on the Knowledge Artifact conceptual model and is general enough to be adopted in different contexts and

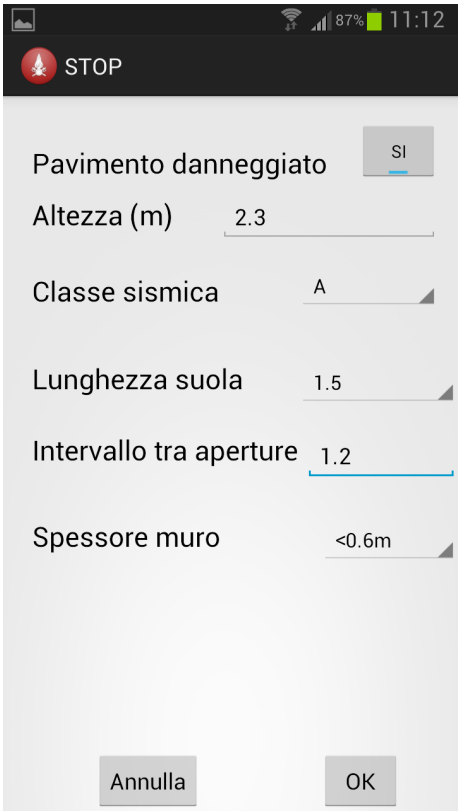

Fig. 4: The STOP–Agent GUI for sending information to the KA–Agent

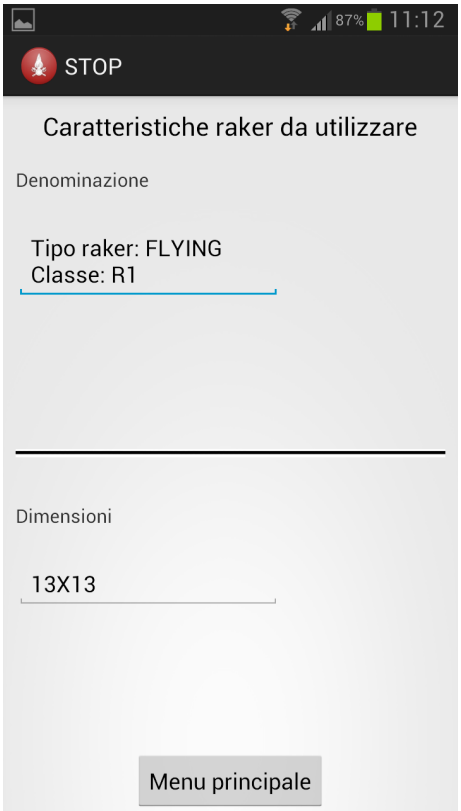

Fig. 5: The KA–Agent sends characteristics on raker to use to the STOP–Agent, in order to make safe a wall

programming paradigms.

In particular, we have integrated it into the AOP model, according to a client–server architecture, to design and implement KBSs remotely exploiting the potentialities of ANDROID OS: in this way, the framework can be executed from every kind of PDAs, like smartphones and tablets, with the possibility to create an ad–hoc KBS when necessary. The framework was applied in a potential collaboration with the Italian Fire Corps and the Civil Protection Department of the Presidency of Council of Ministers, to provide each firemen with tools to understand how to operate in critical situations, like geographically distributed fires and earthquakes management.

The developed system has shown an excellent level of performance, especially about the client side of the project. The ANDROID application requirements are minimal, especially considering the Internet consumption data (an actually critical argument). On the other hand, the memory request by the server is definitely high. This last statement also opens the discussion on future developments which are certainly ample and varied. The server is, without doubt, the component that needs more attention; some modifications that could be made range from the addition of new JESS features to the improvement of memory management and a more efficient management of concurrency.

In the introduction we have already discussed the connection between AOP paradigm and the connection with ANDROID world: an idea to implement the model is to use JADE framework, for several reasons. First of all, JADE framework is a software framework fully implemented in Java language as our native project, and it implements FIPA [2] specifications to support communications and interactions among agents (and we need to model interactions between simple agents a_i and KA-Agent).

More in detail, agents a_i can be programmed using the JADE-ANDROID add-on[8], a JADE module that allows combining the expressiveness of JADE agents communication with the power of ANDROID platform. As already stated, another relevant feature of JADE is the integration with JESS, the rule engine that are the basis on which the expert system is built on according to the KA conceptual model. In fact, KA-Agent has the capability to create the expert system on the basis of domain expert information and it can directly execute it according to data sent it by client. In this way, it could be explored the possibility to import JESS under ANDROID, moving its execution from the current KA–Agent on the server side to the mobile environment.

Concerning the conceptual model of KA–Agent, it should be possible to create a multi-language environment, expanding it to other rule–based languages, such as Drools[9].

Furthermore, the definition of rules could be improved to provide the user with the possibility to define new kinds of constructs, like templates (and the relative slots), functions, shadow facts and so on.

[8]Released with LGPLv2 Licence and available at http://jade.tilab.com/

[9]http://www.jboss.org/drools/

REFERENCES

[1] Y. Shoham, "Agent-oriented programming," *Artificial intelligence*, vol. 60, no. 1, pp. 51–92, 1993.

[2] F. Bellifemine, A. Poggi, and G. Rimassa, "Jade–a fipa-compliant agent framework," in *Proceedings of PAAM*, vol. 99, no. 97-108. London, 1999, p. 33.

[3] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*. Wiley. com, 2007, vol. 8.

[4] A. Ricci, M. Piunti, M. Viroli, and A. Omicini, "Environment programming in cartago," in *Multi-Agent Programming:*. Springer, 2009, pp. 259–288.

[5] M. Ughetti, T. Trucco, and D. Gotta, "Development of agent-based, peer-to-peer mobile applications on android with jade," in *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM'08. The Second International Conference on*. IEEE, 2008, pp. 287–294.

[6] A. Santi, M. Guidi, and A. Ricci, "Jaca-android: An agent-based platform for building smart mobile applications," in *Languages, Methodologies, and Development Tools for Multi-Agent Systems*, ser. Lecture Notes in Computer Science, M. Dastani, A. Fallah Seghrouchni, J. Hbner, and J. Leite, Eds. Springer Berlin Heidelberg, JACA2011, vol. 6822, pp. 95–114. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-22723-3_6

[7] S. Bandini and F. Sartori, "From handicraft prototypes to limited serial productions: Exploiting knowledge artifacts to support the industrial design of high quality products," *AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing)*, vol. 24, no. 1, p. 17, 2010.

[8] C. Brewster and K. O'Hara, "Knowledge representation with ontologies: the present and future," *Intelligent Systems, IEEE*, vol. 19, no. 1, pp. 72–81, 2004.

[9] J. A. Rosen and W. L. Smith, "Influence net modeling with causal strengths: an evolutionary approach," in *Proceedings of the Command and Control Research and Technology Symposium*, 1996, pp. 25–28.

[10] B. Chandrasekaran, T. R. Johnson, and J. W. Smith, "Task-structure analysis for knowledge modeling," *Communications of the ACM*, vol. 35, no. 9, pp. 124–137, 1992.

[11] G. Salazar-Torres, E. Colombo, F. S. C. da Silva, C. A. Noriega, and S. Bandini, "Design issues for knowledge artifacts," *Knowl.-Based Syst.*, vol. 21, no. 8, pp. 856–867, 2008.

[12] G. Schreiber, B. Wielinga, R. de Hoog, H. Akkermans, and W. Van de Velde, "Commonkads: A comprehensive methodology for kbs development," *IEEE expert*, vol. 9, no. 6, pp. 28–37, 1994.

[13] J. Angele, D. Fensel, D. Landes, and R. Studer, "Developing knowledge-based systems with mike," *Automated Software Engineering*, vol. 5, no. 4, pp. 389–418, 1998.