MULTI-AGENT SYSTEM DEVELOPMENT KIT

MAS SOFTWARE TOOL IMPLEMENTING GAIA METHODOLOGY

Vladimir Gorodetski, Oleg Karsaev, Vladimir Samoilov, Victor Konushy, Evgeny Mankov and Alexey Malyshev
St. Petersburg Institute for Informatics and Automation, 14th line, 39,SPIIRAS, St. Petersburg, 199178, Russia
Phone: +7-812-2323570, fax: +7-812-3280685, E-mail:
{gor, ok, samovl, kvg, eman, A.Malyshev }@mail.iias.spb.su,
http://space.iias.spb.su/ai/gorodetski/gorodetski.jsp

Abstract: Recent research in area of multi-agent technology attracted a growing attention of both scientific community and industrial companies. This attention is stipulated by powerful capabilities of multi-agent technology allowing to create large scale distributed intelligent systems, and, on the other hand, by practical needs of industrial companies to possess an advanced and reliable technology for solving of practically important problems. Currently one of the topmost questions of the research is development of powerful methodologies for engineering of agent-based systems and development of more effective and efficient tools supporting implementation of applied systems. The paper presents one of such tools, Multi Agent System Development Kit, based on and implementing of Gaia methodology. It supports the whole life cycle of multi-agent system development and maintains integrity of solutions produced at different stages of the development process.

Key words: Software engineering, Multi agent systems, Methodology, Software tool

1. INTRODUCTION

Although agent-oriented software engineering is being a subject of active research for over a decade, it does not still come to the age of maturity required to be rated as an industrial technology. In spite of rich theoretical achievements in this area, there practically exist no powerful Multi-Agent System (MAS) software tools capable to support the whole life cycle of industrial MAS comprising analysis, design, implementation, deployment and maintenance, although to date a lot of MAS software tools are developed. Among them, the most mature and popular are AgentBuilder [18], Jack [16], JADE [3], ZEUS [7], FIPA-OS [11], agentTool [9], etc.

Analysis of the existing software tools allows understanding potential directions that can bring necessary results for considerable increasing powerfulness and maturity of MAS software tools. One of them is exploitation and adaptation of the experience accumulated within object–oriented approach and existing (at least, de-facto) "standards" of analysis, design and implementation commonly used in the information technologies [5]. In this respect, several initiatives are currently undertaken, and one of them is Agent UML project [1] that is being carried out by two leading international organizations, FIPA and OMG, focused on standardization within advanced information technology scope.

The other source of evolution and perfection of the MAS software tool is further development of the existing methodologies of agent–based system engineering that should potentially provide designers with new opportunities. Among the existing methodologies, Gaia [19], MESSAGE [6], MaSE [10], Prometheus [17], Adelfe [4], Tropos [12] and some others pretend to be very promising. Gaia methodology considers two stages of applied MAS development that are (1) analysis and (2) design. The objective of the analysis is to reach "*an understanding of the system and its structure (without reference to any implementation details)*" [19]. This stage assumes design of solutions of a high-level of abstraction concerning system organization, i.e. discovery MAS tasks, discovery roles and their responsibilities within particular MAS applications, description of roles' tasks and high-level scheme of roles' interactions during fulfillment of the MAS tasks, etc. The objective of the design stage is "*to transform the abstract models derived during the analysis stage into models at a sufficiently low level of abstraction that can be easily implemented*" [19]. This stage assumes formal specification of how the community of agents interacts in order to solve the MAS tasks and also formal specification of each particular agent of applied MAS under development.

Thus, the current trends in MAS technology prompt that sound methodology enriched by ideas and experience from object–oriented design scope can considerably improve MAS technology providing it with such properties as consistency and integrity of solutions being produced at the subsequent stages of MAS engineering life cycle.

The paper presents a recently developed software tool, Multi-agent System Development Kit 3.0 (MASDK) providing support for the whole life cycle of applied MAS development. This software tool is based on Gaia methodology integrated with ideas of object–oriented design of MAS resulted from the Agent UML project. This software tool was the subject of research during last years and the current version is the third one. Previous versions of MASDK [13] were used for rapid prototyping of different MAS applications [14, 15]. The experience accumulated so far allowed understanding of the drawbacks and limitations of the previous two versions of MASDK and developing adequate requirements to its current version, 3.0.

MASDK 3.0 software tool is now being evaluated based on design of applied MAS for detection of intrusions in computer network, situation assessment and design activity support. In the rest of the paper, section 2 outlines general ideas of MASDK supported technology. Sections 3, 4 and 5 describe the analysis, design, and implementation stages respectively of MAS technology. Conclusion outlines the paper results and future work.

2. OUTLINE OF MASDK 3.0 SOFTWARE TOOL AND TECHNOLOGY SUPPORTED

MASDK 3.0 software tool consists of the following components (Fig.1): (1) *system kernel* which is a data structure for XML–based representation of applied MAS formal specification; (2) *integrated* set of the user friendly *editors* supporting user's activity aiming at formal specification of an applied MAS under development at the analysis, design and implementation stages; (3) library of C++ classes of reusable agent components constituting what is usually called *Generic agent*; (4) communication platform to be installed in particular computers of a network; and (5) builder of software agent instances responsible for generation of C++ source code and executable code of software agents as well as deployment of software agents over already installed communication platform.

Specification of applied MAS in the system kernel is being carried out by use of editors structured in three levels. Editors of the *first one* provide support for meta–level specification of applied MAS corresponding to the

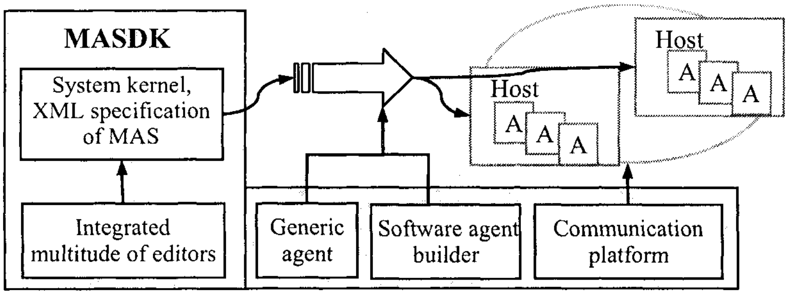


Figure 1. MASDK software tool components and their interaction

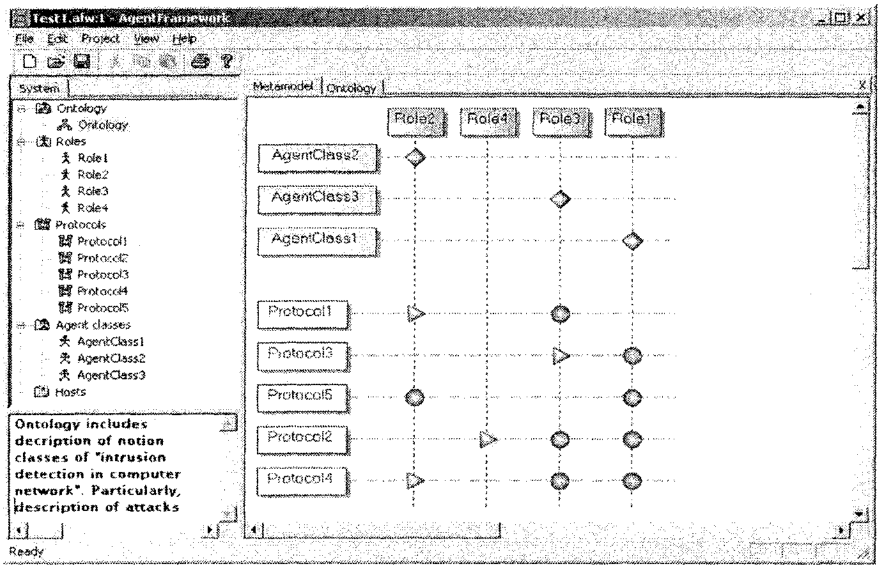
analysis stage in Gaia methodology. The set of the first level editors includes (1) *application ontology* editor, (2) editor for *description of roles, names of agent classes, and high-level schemes of roles' interactions,* (3) editor of *roles' interaction protocols.* Editors of the *second level* support the design activities and primarily aim at specification of agent classes. The following editors are used for this purpose: (1) editor specifying *meta-model of agent classes' behavior*; (2) editor specifying *particular agent functions and behavior scenarios* in terms of state machines; (3) editor specifying software *agent private ontology* inheriting the notions of shared domain ontology.

Editors of the *third level* support implementation stage of applied MAS and aim at (1) implementing (in C++ language) a *set of particular components and functions* specified in design stage; (2) specifying *sub-network* within which the designed MAS is to be deployed; (3) specifying *lists of agents instances of all classes* with the references to their locations (hosts names), and (4) initial states of *mental models* of each agent instance.

Applied MAS specification produced by designers exploiting the above editors is stored as XML file in the system kernel. This specification, including set of particular components and functions implemented in C++, and *Generic Agent* reusable component form the input of the software agent builder generating automatically software code based on *XSLT* technology.

3. ANALYSIS STAGE: APPLIED MAS META-MODEL

At the analysis stage conceptual description of applied MAS is produced. While having the MAS high–level tasks determined, the first step of problem domain analysis assumes (1) discovery and description of roles, (2) high–level mapping of role interactivities to protocol list, (3) determination of agent classes' names, and (4) designation of roles to them. At that, information about roles, interaction protocols, their mapping and textual description of the roles behavior according to respective protocols are used for defining the set of agent classes and assigning of roles to them. It is assumed that each agent class can perform one or several roles. Actually, this mapping determines functionalities of agent classes. Altogether these descriptions constitute what is called hereinafter an applied MAS meta–model. This activity is supported by editor (it is opened within main MASDK window) called "*Agent Framework*" (Fig.2). Project browser (in the left-upper area) and (2) description of element selected in browser (in the left-down area) are two other sections of the framework.



Ontology includes decription of notion classes of "intrusion detection in computer network". Particularly, description of attacks

Figure 2. Meta-model editor

Detailed description of the roles interaction protocols is one of the key tasks of the analysis stage. Realizing the importance of this task was perhaps one of the reasons motivated the Agent UML project [1] initiated by both FIPA and OMG. Let us remind that the project objective is an extension of UML language to agent–based system specification language, and one of the focuses of this Project is development of a language directly destined for specification of agent interaction protocols [2].

MASDK 3.0 includes graphical editor of roles' interaction protocols that makes use of main principal solutions of Agent UML project. Not all the proposals of Agent UML project, pretending to develop future standards of MAS, are used in MASDK 3.0, because Agent UML is currently in progress and, thus, some of its proposals are tentative while others seem to be not well grounded or too overloaded with secondary notations, what makes it difficult both to understand and to implement such notations. For example, agent interaction protocol specification language used in MASDK 3.0 is a simplified version of the analogous language of Agent UML although the former preserves basically the expressive power of the latter.

4. DESIGN STAGE: AGENT CLASSES

Applied MAS meta-model developed at the analysis stage is further used as an input of the design stage. Specification of agent classes is a key point of this stage. Generalized architecture of an agent is presented in Fig.3. Its basic components are the followings: (1) invariant (reusable) component called *Generic Agent*, (2) meta-model of agent class's behavior, (3) a multitude of functions of agent class represented in terms of state machines; (4) library of specific auxiliary functions. In the first component, the common agent meta–behavior rules are specified. They constitute the invarint component of any software agent and particular agent class design is focused on specification of its three other components.

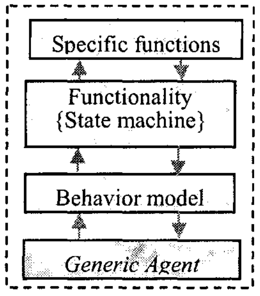


Figure 3. Agent structure

Specification of agent class' behavior meta-model is supported by editor depicted in Fig.4. It supports transformation of the conceptual solutions into formally specified structure of components for each agent class. Its main component is *Agent class functionality model.* It includes description of agent class' functions list. The initial information about each agent class consists of textual descriptions of the respective roles and tasks of roles (defined at analysis stage of MAS development) allocated for execution to agent class. Detailed specification of the above functions is the designer's responsibility. An example of the set of such functions represented as state machines is demonstrated graphically in Fig.4.

The rest of the agent class components can be divided into two groups. The first of them specifies event classes initiating execution of functions specified within the following four components:

Input messages. This component indicates relations between certain protocols and functions. The sense of these relations is to point out the protocols in which agent class in question takes part if the above protocols are initiated by certain other agent classes. These protocols are determined formally by use of meta-model of applied MAS developed at the analysis stage. Let us note that the first messages of protocols play the roles of events initiating execution of the respective functions.

User commands. This component is present in agent class if it interacts with user initiating certain agent class behavior in certain situations. In this component, if any, the user commands mapped to respective functions are specified.

Pro-active model. If non-empty, it includes specification of rules in the form *"When ... if ... then ...".* The precondition "*When*" indicates events (e.g., time instants) when the second condition has to be the subject of checking. The condition "*if*" specifies the agent class mental state when the function indicated in the third part of the rule has to be executed.

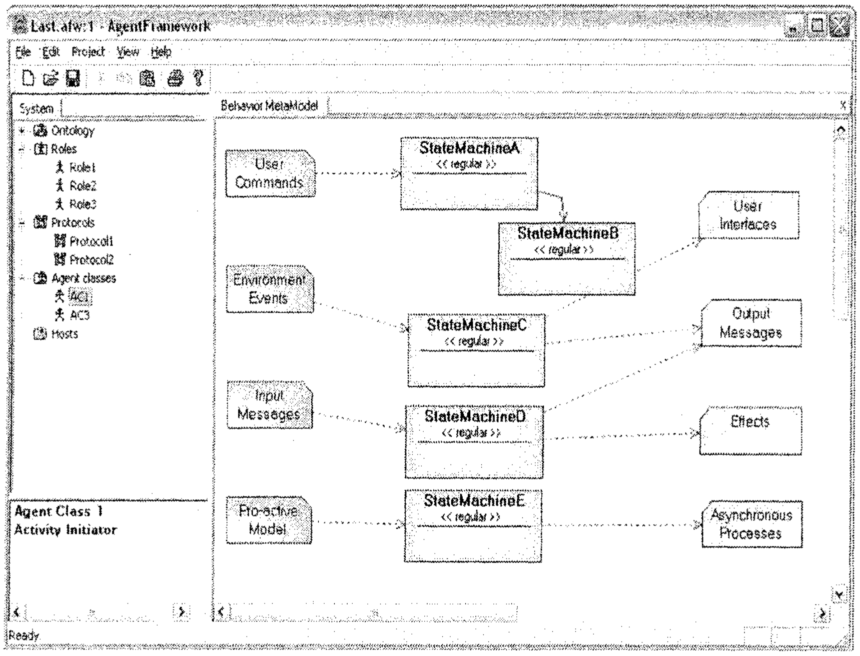


Figure 4. Meta-model of agent class behaviour

Environment events. Content of this component specifies classes of environmental events and respective behavior of agent class.

The second group of components describes actions that can be or has to be performed during execution of respective functions. They are 1) initiation of interaction protocols, 2) dialogs with user, 3) effects on environment and 4) execution of specific functions in asynchronous mode. In contrast to the components of the first group, which must be completely specified at this stage, specification of the components of second group is considered as specific application-dependent requirements to be taken into account in the subsequent steps of the design stage. The editor supports graphical mode of representation of the relationships between components and functions in form of explicit connections between them. Connections between functions represent the facts that certain functions are nested ones and their execution is invoked by other functions.

Each function is represented (specified formally) in terms of state machine composed of the following standard components: (1) state machine states; (2) transitions and conditions determining the transition selection depending on the agent current state; (3) state machine behavior corresponding to each particular state. Specification of the first and the second components is executed at the design stage and it is supported by graphical editor. On the contrary, the third component is specified at the implementation stage. Specification of state machines is carried out in two steps. The first step is carried out automatically based on meta-models of agent classes' behavior. The second step in carried out by designer developing the first step specification through insertion of new states with respective updating of the transitions structure.

5. IMPLEMENTATION AND DEPLOYMENT

The next stage of an applied MAS development corresponds to its implementation that is completely based on the results of two previous stages. Implementation technology consists of the following activities:
1) Implementation of the private ontology of each agent class, which inherits the shared application ontology. The inherited notions of shared ontology are used in specification of agent class messages content. The rest of notions specified in the private ontology of agent class (they can also inherit the notions of shared ontology) represent the agent class mental model.
2) Implementation of the library of specific auxiliary C++ classes of all the agent class components specified at the design stage. These classes correspond to scripts of agent classes behavior in particular states of state machines and functions (see Fig.3).
3) Specification of the applied MAS configuration, in particular, specification of instances of each agent class and indication of their locations in computer network where applied MAS under development is deployed.
4) Specification of initial mental model of each agent.

The next step of implementation stage is destined for generation of software agent instances and their placing in the respective hosts of computer network according to their addresses. While generating software code of an agent instance, three types of its components are combined. These components are (1) reusable component called *Generic agent*; (2) C++ library that is software implementation of particular functions; and (3) software components implementing meta-models of behavior and state machines of agent classes. It should be noted that software code of the last component and agents as a whole are generated automatically.

6. CONCLUSION

MASDK environment described in the paper possesses a number of practically important advantages allowing noticeably decrease the total amount of efforts and costs of multi–agent systems development. Among them, the most important ones are the followings:

1) Development process is carried out according to a well grounded methodology that is the Gaia methodology, whose abstract notion classes closely interrelate with the other ones used at the subsequent stages of applied MAS design and implementation.

2) User friendly multitude of graphical editors of MASDK provides clear presentation and simple understanding of all the stages of applied MAS development process. Together with the property mentioned in the item 1, graphical mode of the development process provides simple and clear cooperation between designers and programmers during the whole life cycle of applied MAS.

3) Due to well structured set of abstract notion classes, the MASDK environment provides development process with a capability of consistency maintenance and checking of integrity at all stages of process.

4) Representation of interaction protocols that is one of the key tasks of agent-based systems specification, based on solutions evolving experience of object-oriented approach makes the resulting applied MAS potentially compatible with the de-facto standards that are being developed within OMG and FIPA efforts within Agent UML project.

5) The developed technology is strongly based on reusability idea and *Generic agent* library, which includes necessary set of reusable solutions/software components, allows reducing the development process to specification of application-oriented knowledge.

Due to these advantages MASDK environment pretends to be a sufficiently effective and efficient tool with industry-oriented potential.

Currently the basis components of MASDK software tool are implemented and it is being validated via its use for development of applied MAS in such application domains as data fusion, computer network security, and design activity support and monitoring.

ACKNOWLEDGEMENT

This research is supported by grants of European Office of Aerospace R&D (Project #1993P) and Russian Academy of Sciences (Department of Information Technologies and Computer Systems), Project # 4.3.

REFERENCES

1. Agent UML: http://www.auml.org/

2. Bauer, B., Muller, J. P., Odell, J.: Agent UML: A Formalism for Specifying Multiagent Interaction. In: Ciancarini, P. and Wooldridge, M. (eds): Agent-Oriented Software Engineering, Springer-Verlag, Berlin, (2001) 91-103
3. Bellifemine, F., Caire, G., Trucco, T., Rimassa, G.: Jade Programmer's Guid. JADE 2.5 (2002) http://sharon.cselt.it/projects/jade/
4. Bernon, C., Gleizes, M.P., Peyruqueou, S., Picard, G.: Adelfe, a methodology for Adaptive Multi-Agent Systems Engineering. In: Third International Workshop "Engineering Societies in the Agents World" (ESAW-2002), Madrid, (2002)
5. Booch, G.: Object-Oriented Analysis and Design, 2nd ed., Addison-Wesley: Reading, MA, (1994)
6. Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez, J., Pavon, J., Kearney, P., Stark, J., and Massonet, P.: Agent-oriented analysis using MESSAGE/UML. In: Wooldridge, M., Ciancarini, P., and Weiss, G., (editors): Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001), (2001) 101-108
7. Collis, J. and Ndumu, D.: Zeus Technical Manual. Intelligent Systems Research Group, BT Labs. British Telecommunications. (1999)
8. Dam, K. H., and Winikoff, M.: Comparing Agent-Oriented Methodologies. http://grial.uc3m.es/~dcamacho/resources/papersAOSE/dam03comparing.pdf
9. 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, LNCS. Vol.1986, Springer Verlag, (2001)
10. DeLoach, S. A., Wood, M. F., and Sparkman, C. H.: Multiagent systems engineering. In: International Journal of Software Engineering and Knowledge Engineering, 11(3), (2001) 231-258
11. FIPA-OS: A component-based toolkit enabling rapid development of FIPA compliant agents. http://fipa-os.sourceforge.net/
12. Giunchiglia, F., Mylopoulos, J., and Perini, A.: The Tropos software development methodology: Processes, Models and Diagrams. In: Third International Workshop on Agent-Oriented Software Engineering, Jula (2002)
13. Gorodetski, V., Karsaev, O., Kotenko, I., Khabalov, A.: Software Development Kit for Multi-agent Systems Design and Implementation. In: Dunin-Keplicz, B., Navareski, E. (Eds.): From Theory to Practice in Multi-agent Systems. Lecture Notes in Artificial Intelligence, Vol. # 2296, (2002) 121-130
14. Gorodetski, V., Karsaev, O., Konushi, V.: Multi-Agent System for Resource Allocation and Schedulling. In: Lecture Notes in Artificial Intelligence, Vol. # 2691, (2003) 226-235
15. Gorodetsky, V., Karsaev, O., Samoilov, V.: Multi-agent Technology for Distributed Data Mining and Classification. In: Proceedings of the IEEE Conference Intelligent Agent Technology (IAT-03), Halifax, Canada, (2003) 438-441
16. Jack. Jack intelligent agents – version 3.1, agent oriented software pty. Ltd., Australia, http://www.agent-software.com.au .
17. Padgham, L. and Winikoff, M.: Prometheus: A pragmatic methodology for engineering intelligent agents. In: Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, Seattle, (2002) 97-108
18. Reticular Systems Inc: AgentBuilder An Integrated Toolkit for Constructing Intelligent Software Agents. Revision 1.3. (1999) http://www.agentbuilder.com/.
19. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. In: Journal of Autonomous Agents and Multi-Agent Systems, Vol.3. No. 3 (2000) 285-312