

The Reuse Policy in Developing Multi-agent System

Xue Xiao

Lab of Intelligence Science
Henan Polytechnic University
Jiaozuo, Henan, P.R.China
Email: jzxuexiao@126.com

Sun Zhi

The College of Tourism and Management
Zhe Jiang Forestry University
Lin an, Zhejiang, P.R.China
Email: curelandscaper@tom.com

Luo JunWei

Lab of Intelligence Science
Henan Polytechnic University
Jiaozuo, Henan, P.R.China
Email: ljwonly@yahoo.com.cn

Abstract—The agent-oriented (AO) methodology is an effective means for constructing distributed systems. Despite a great deal of research, a number of challenges still exist before making agent-based computing a widely accepted paradigm in software engineering practice. In order to solve the problem of “a variety in number, difficult to apply”, the paper presents a hierarchical development architecture (HDA) for customizing a new AO methodology according to the given project. Through applying the HDA-based meta models and design patterns, developers can build applications from third party off-the-shelf solution components. To exemplify its feasibility and effectiveness, the construction of C4I system is presented as a case study.

Index Terms—design pattern, customization, HDA

I. INTRODUCTION

Agents and multi-agent systems (MASs) have recently emerged as a powerful technology to face the complexity of a variety of today’s ICT (information and communication technology) scenarios, such as distributed system, web service and so on. What’s more, the emergent general understanding is that MASs, more than an effective technology, represent indeed a novel general-purpose paradigm for software development [1]: agent-based computing can promote designing and developing applications in terms of agents, that are situated in an environment and can flexibly achieve their goals by interacting with one another in terms of high-level protocols and languages.

In this sense, MASs offer a new and often more appropriate route to the development of complex computational systems, especially in open and dynamic environments. Therefore, in the last few years, there has been a great deal of research related to the identification and definition of suitable models and techniques to support the development of distributed systems in terms of MASs [2], such as formal modeling approaches, development methodologies and modeling techniques, specifically suited to the agent-oriented paradigm. However, despite the great deal of research in the area, agent oriented software engineering (AOSE) is still a relative young area. Currently, there are, as yet, not standard methodologies, development tools, or software architectures. There still exist a number of challenges before making agent-based computing a widely accepted paradigm in software engineering practice [3]. The problem of “a variety in number, difficult to apply” has become an obstacle in turning agent-oriented software abstractions into practical tools for facing the complexity of modern application

areas.

In order to handle the challenge, we propose a hierarchical development architecture(HDA) in the paper. Based on the HDA, developers can combine different meta models originated from various AO methods to customize a best suited development methodology for the given project. The HDA-based design patterns bridge the gap between design abstraction and software implementation, which help developers to build MASs efficiently. The rest of the paper is organized as follow: Section 2 introduces the hierarchical development architecture in detail. In section 3, a research project on the construction of C4I system exemplifies the effectiveness of HDA as a case. Some conclusions are presented in section 4.

II. HIERARCHICAL DEVELOPMENT ARCHITECTURE(HDA)

A. Current problems in AOSE

At the moment, a lot of attempts have been made in AOSE, such as the building of MASs and the definition of suitable models, methodologies and tools for these systems. In AOSE, the notion of an autonomous agent is used as the fundamental modeling abstractions to map real-world items to computational model. Thus, abstract problem solutions can be effectively implemented, and software complexity can be decreased. Nevertheless, the research of AOSE is still in its early stages, and a lot of challenges need to be handled before AOSE can deliver its promises, becoming a widely accepted and practically usable paradigm for the development of complex software systems. There is a current lack of mature agent-based software development methodologies. This deficiency has been pointed out as one of the main barriers to the large-scale uptake of agent technology[3].

In the area of AO methodologies, researchers from different fields give emphasis to the different aspects of the process. In order to pursue generality, some AO methods take agent as a modeling unit and only emphasize agents’ external characteristics and their relationships between each other, e.g. the Gaia methodology [20]. On the contrary, the other AO methods are bound to some type of concrete agent architecture to facilitate software implementation, e.g. the AAIL methodology [4]. Some research works (e.g. [5],[6]) provide comparison studies between different AO methodologies, showing that each AO method has its own weaknesses and strengths. Many

users had trouble in finding a method that would satisfy their needs completely.

It's often infeasible or inefficient to use a single kind of AO method to solve all the problems in the construction of MASs. As a matter of fact the need for systematic principles to develop situation-specific methods, perceived almost from the beginning by the object-oriented community, has led to the emergence of the method engineering approach. In the last years, the method engineering approach is proved successful in developing object oriented information systems [7]. Its importance in the object-oriented context should be evaluated considering not only the direct influence (not so many companies and individuals work in this specific way) but mainly the indirect consequence. The most important and diffused development process (e.g., the Rational Unified Process [8]) is in fact not rigid, instead being a kind of framework within which the designer can choose his/her own path.

We believe that the agent-oriented community should follow a similar path, trying to adapt the method engineering for using it in agent-oriented design[9]. Some researchers [10, 11, 12] have proposed that the AOSE methodologies should be created and customized in a modular way, which enables developers to build project-specific methodologies from meta models, just like applications built from reusable off-the-shelf components.

Our aim is to propose an open architecture, which can guide developer to assemble a new methodology tailored to the given project by fitting AOSE meta models in appropriate position. Thus, each new project has its own methodology, built up from components of other methodologies. As a result, the advantages of different AO methods can be taken of and their drawbacks can be overcome. Furthermore, such exploitation speeds up development, avoids reinventing the wheel, and enables sufficient time to be devoted to the value added by the multi-agent paradigm.

B. The definition of HDA

Against the background, a hierarchical development architecture (HDA) for customizing AO methodology is proposed in figure 1. The architecture consists of five phases, covering the whole development lifecycle to establish a systematic development method. The requirement analysis phase defines development goal, which lays foundation for the following development. The MAS architecture phase represents the outline of the system configuration and organizational behaviors and is not dependent on any specific agent platforms. The agent modeling phase depicts all components of each agent specialized in agent structure. The software implementation phase gives the detail of the system configuration and the program codes implemented according to the design of the above phases. Finally, the verification phase is used to ensure that the software to be constructed can meet the demand of users.

In the architecture, each phase is categorized into a layered model structure further. Based on the goal and the task of each layer, user can fill meta models into the appropriate layers to handle different kinds of quality attributes. Instead

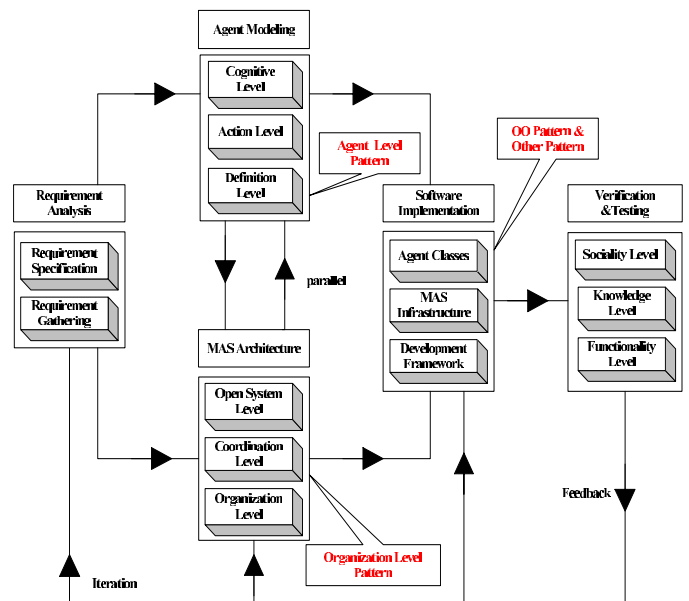


Fig. 1. The Hierarchical Development Architecture(HDA)

of creating incompatible techniques, models and CASE tools for each methodology, modular and reusable meta models can be created once, and shared within different methodologies. It represents significant savings in development cost and learning cost. In application of HDA, developers need to follow these steps:

(1) Factor the overall problem and identify what layers should be selected in HDA for a specific application. Based on this, developers can select existing AO methodologies to extract appropriate meta models.

(2) Fill corresponding meta models into appropriate layers and form a new AO method through tailoring and combination on particular project. The meta model which belongs to some layer should facilitate achieving the goal of the layer. Otherwise, it's not appropriate to put it into the layer.

(3) In order to meet the demand of customizing a new approach, developers need to enforce and modify the extracted meta models.

(4) The increasing details of a new approach to be constructed are developed step by step. Together, the layers enable the configuration of a new AO approach that can be appropriate for specific application.

(5) Once the methodology is composed, the designers will perform the established process obtaining a model of the system - an instantiation of the MAS meta-model - that solves their problem.

The layered model accords with the pay-as-you-go philosophy : programmers do not need to use all the layers provided by the architecture. Layers that are not used do not require programmers to know anything about them, neither add any computational overhead. Thus, developers can quickly grasp the most important concepts in the development of MASs. For example, if the system needs to run in open environment, then a methodology that incorporates "open system layer" can

be adopted in “MAS architecture phase”. If some specific agent platform need to be adopted in another part, choose a methodology that supports it in “Software implementation phase”. The details of each layer are given below:

- **The requirement analysis phase** : i) Requirement Gathering Layer: gather, identify and define requirements according to practical application, including functional and non-functional. ii) Requirement Specification Layer: the requirement statements will be mapped to a MAS, i.e. the requirements are specified in terms of agent, role and organization.

- **The MAS architecture phase** : i) Organization Layer: define how multi-agents construct the whole system and realize the required functions. The whole system is conceived in terms of an organized society in which each agent plays specific role and interacts with other agents according to protocols determined by the roles of the involved agents. ii) Coordination Layer: Based on the interaction among the constitute agents, the component agents can be explicitly designed to cooperatively achieve a given goal. iii) Open System Layer: As a more general case, agents might not be co-designed to share a common goal, and have been possibly developed by different people to achieve different objectives. Moreover, the composition of the system can dynamically vary as agents enter and leave the system.

- **The agent modeling phase** : i) Definition Layer: make clear some basic aspects of each agent: what to think of? what resources to own? what task to shoulder? What acquaintance to interact? ii) Action Layer: how to think and form a plan? iii) Cognitive Layer: how to enforce the plan and react outside?

- **The software implementation phase** : i) Agent Society Layer: the layer can be reused, meaning that it is independent of underlying mechanism. ii) MAS Infrastructure Layer: provides some fundamental services. In MAS infrastructure layer, if we apply JADE or other AO infrastructure, we must use other models that comply with the products provided by this specific infrastructure. iii) Development Framework Layer: depends on the implementation platform of our choice, including development tools and programming language. If we use Java, we may use UML Class Diagrams for this last abstraction level in agent society layer.

- **The verification and Testing phase** : i) Functional level: ensure that system is stable, reliable, effective and meet functional requirements. ii) Knowledge Level: single agent can reason, learn and be adaptive to environment. iii) Sociality Level: multi-agent system can meet the demand for macro-level performance.

C. The pattern-driven development policy

In the application of HDA, each successive pass will add additional detail and a series of system design artifacts is produced eventually, i.e. an ordered sequence of steps, an identifiable set of models, and an indication of the interrelationships between the models, showing how and when to exploit which models and abstractions in the development of a MAS. The HDA accords with a “top-down” development policy, which focuses on design structure of the whole multi-agent

system. However, the implementation details of each model are disregarded in HDA, which may lead developers without background in agent technology to be confused about how to turn design model into software implementation. What’s more, this results in rediscovering solutions to common design problems without benefiting from how they were resolved in the past.

Therefore, we need to introduce a kind of “bottom-up” development mechanism to complement the defects of goal-driven development process. A suited choice is to build an agent system incrementally from well-documented agent patterns. Recently, design patterns are attracting attention in usual software development represented by object-oriented software. Design patterns are explicit formulations of good past software development experiences consisting of proven solutions to recurring problems that arise within some contexts and systems of forces. The patterns realize easy reuse of good software design. These concepts are becoming indispensable in developing practical large-scale software in a low cost by promoting reuse. Individual patterns can, furthermore, be linked to each other in the form of pattern languages, which guide the designer through the design process.

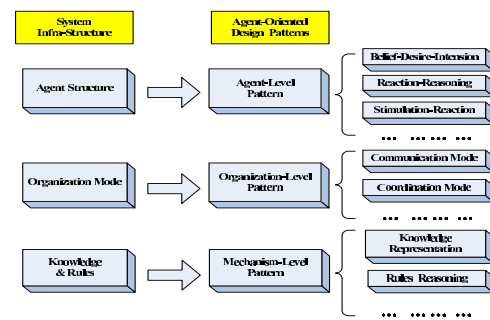


Fig. 2. The Pattern Driven Development

Against this background, we describe a pattern-driven design process that complements goal-driven design approaches in figure 2. As fundamental element of constructing method[13, 14, 15, 16], each meta model occupies a position in the HDA and plays a key role in relating macro design to micro detail. Generally speaking, after determining the scope and the purpose of the existing methodologies, the meta model is created by isolating general-purpose common features from them. Based on the development tasks and quality goals, the methodology is partitioned into multiple meta models and the dependencies between the new models and any existing models are determined. The remaining parts of the methodology are then componentized into special purpose “value-adding” features.

An interaction meta model extracted from Gaia is shown as an example in figure 3, a comprehensive list of potential elements included in meta model are shown:

D. The HDA-based design pattern

Currently, some researchers have begun to apply design patterns to MASs development. Generally speaking, there are

Interaction Meta Model	
Parent	Gaia Methodology
Pre-Condition	Role Model, Organization Structure Model
Description	The model captures the dependencies and relationships between the various organization roles in terms of one protocol definition for each type of inter-role interaction.
Solution	In Gaia, a protocol can be viewed as an institutionalized pattern of interaction, which has been formally defined and abstracted away from any particular sequence of execution steps. It focuses attention on the essential nature and purpose of the interaction, rather than on the precise ordering of particular message exchanges.
Implementation	Meeting Pattern, Ambassador Pattern, Finder Pattern Proxy Pattern, Badges Pattern, White Board Pattern Translator Pattern
Support	FIPA KQML Language, ACL Language
Consequence	Once agents can communicate, the level of abstraction can be raised to the co-ordination level, wherein bargaining and negotiating is possible.

Fig. 3. The Interaction Meta Model

four kind of sources : (i) existing excellent AO methodologies, which can guide us which kind of pattern is demanded. (ii) some excellent agent-based practices and agent platform, such as JADE, ZEUS and so on. (iii) OO design pattern experience, including design, coding and analysis.

However, only a coarse-grain development route (analysis - design - implementation - testing - release) is provided to integrate all meta models. To a large degree, whether the application of patterns is effective lies on developer's experience. With the number of the patterns increasing, it's becoming more and more difficult to select and apply appropriate design pattern.

As a platform for sorting and managing those design patterns, the HDA gives an explicit guide about how to apply those patterns and an order of examination of pattern application. Each pattern occupies a position in the HDA, in which each pattern contributes to the completion of patterns "preceding" it in the architecture, and is completed by patterns "succeeding" it.

As shown in figure 2, according to the HDA phases and specific agent platforms, the patterns are classified into four categories:

Organization Level Pattern : This kind of pattern is applied in the development of the MAS architecture, which consists of inter-organization pattern(the relationship between organizations) and inner-organization pattern(the relationship between agents). Two types of patterns are only different in basic unit: one is an organization, and the other is an agent. The type of pattern is not dependent on any specific agent platforms and can be easily reused.

Agent Level Pattern : This kind of pattern is applied in the development of an agent unit. A set of patterns is used to describe building blocks from which an agent can be built.

The patterns should be selected as they are specialized in the individual agent platforms and make good use of the advantage of the platforms.

Object-Oriented Design Pattern : This kind of pattern is the usual ones for implementation using OO languages such as Java and C++. One of the representative groups of such patterns was investigated by the Gang of Four (GOF).

Others : Many patterns cut across all boundaries.

Patterns need to capture hidden structure and decisions: deep components of architecture and design which are larger than any architectural building block such as procedures and objects. The task of extracting design pattern is a long-term and tough work, which needs to summarize expert experience and iterate according to feedback over and again. In order to make pattern easy to read and understand, it's important to determine your own style/way of documenting your pattern. Here gives an example of coordination-layer pattern in figure 4 to explain the referenced format of agent design pattern :

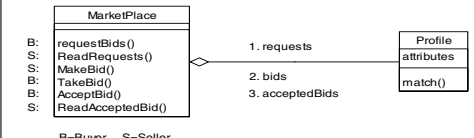
Coordination -Layer Design Pattern	
Forces	<ol style="list-style-type: none"> partitioning of work can be used to increase reliability, performance or accuracy. Partitioning should be transparent to clients. Agents need to collaborate to perform complex tasks. Agent behaviors may conflict. Agents and agent applications should be evolvable.
Name	Marketplace
Problem	how do agents exchange location-specific data with other agents?
Context	Your agent system evolves constantly. Instead of setting up links between agents up-front you let the agents create them on-the-fly.
Solution	
Consequence	Buyers choose the product that "best" meets the attributes they care most about (least expensive, highest quality).
Related Patterns	Once buyers have located appropriate sellers, they may still need to negotiate the terms of transaction (such as the delivery conditions). This leads to the Negotiating Agents Pattern.
Example	The first examples is the Contract Net protocol pioneered by Smith. The MAGNET market infrastructure provides support for a variety of transactions, from simple buying and selling to multi-contract negotiations

Fig. 4. The market design pattern

III. APPLICATION OF HDA

A. Project description

The C4I (command, control, communication, computer and Information) system is the core of the whole naval warship, which is used as information process (including collection, transformation, process and transmission), fighting support and weapon control. C4I system integrates different kinds of weapons together to ensure that the integral performance can be improved to a high degree. In order to achieve the

goal, three main problems need to be considered during the construction of C4I system: i) how to harmonize a number of components which may have been developed by teams having worked separately on different portions of the system; ii) how to adopt new technology to deal with multiple, heterogeneous and even dynamic application environments; iii) how to integrate many different technologies, languages, paradigms and legacy systems together in an effective and fruitful way. Apart from the (problem-solving) functionality, the C4I system must also satisfy properties such as reliability, fault-tolerance, maintainability, scalability etc.

As an example of the distributed problem solving system, the components of C4I system are devoted to controlling and directing different phases of the physical process: information collection - information process - fighting support - command - weapon control. It leads to a conception of the whole system as a number of different MAS organizations, one for each phase of the process. Each organization can be viewed as being made up of agents being involved in the process. Different agents are delegated different task which contribute to the accomplishment of system goal.

Therefore, we apply agent technology as an original and more effective way to solve highly-complex problems in the construction of C4I system. Through analysis and verification over and again, we decided to derive a new AO approach from HDA to solve the problems in the construction of C4I system on naval warship. We hope that the new approach can take the best of the different approaches and overcome their respective shortcomings. What's more, it should take advantage of the existing agent platforms and resources as more as possible, such as JADE.

B. Methodology customization

The C4I system is closed, all component agents are supposed to be innately cooperative to each other and they can trust one another during interactions. The internal structure of agent should support the external performance. The usual agent system development methods cannot give guidelines that are easy to understand. In order to facilitate the engineering change in AOSE, the new approach needs to be associated with patterns closely. The whole design step can be effectively implemented, which provides the analyst and designer with a way of how to go from theoretical definitions to practical implementation elements.

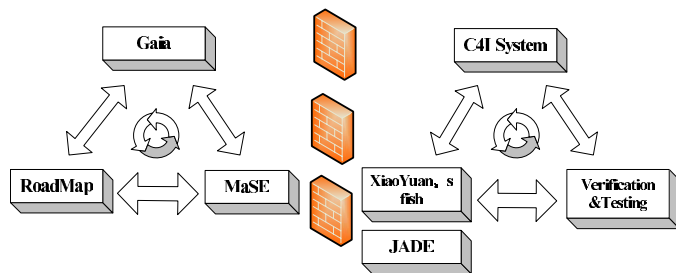


Fig. 5. The New Methodology from HDA

Aimed at those characteristics, we merged several representative AO methodologies: RoadMap[17], Gaia[18], MaSE[19], and Xiaoyuan's fish model[20], which is shown in figure 5. In selecting appropriate AO methods for customization, specifically we consider: i) Agent structure: this means how each of the meta-models represents the agent and its most common elements. ii) Agent interactions: agents of different meta-models are supposed to interact using communications or the environment. Communications are sometimes specified by attributes like interaction protocols, content language and so on. iii) Agent society and organizational structure: the goal of some of these meta models is to model a specific society or an organizational infrastructure constrained by rules that enforce agents to some collective or individual behavior. iv) Agent implementation: the code-level structure of the agent system.

The Gaia meta models are mostly devoted to represent a MAS system as a social organization. In RoadMap meta models, a great effort is made on requirement phase in order to complement the defects of the Gaia. The MaSE meta models aim to conciliate classical OO software engineering concepts with the potentiality of the agent-based approach while pursuing the goal of a traceability of the solution from requirements to the related code implementation. Xiaoyuan's fish model focuses on displaying the cognitive and behavior issues. Because of limited space, we will have to diffusely discuss the details about combination in other papers: how to extract appropriate meta models from each method, how to attach design patterns to meta model, how to fill meta models into the HDA, and how to derive a new approach from the HDA.

Through applying the method, a large-scale complex C4I system is decomposed and reconstructed successfully. Considering physical deployment and communication overload, the whole C4I system is decomposed into five agent organizations. Each of them bears different responsibility: monitoring enemy (Information Collection Organization, ICO); filtering and processing the collected information (Information Process Organization, IPO); advising (Decision Supporting Organization, DSO) and making decision (Decision Making Organization, DMO); attacking enemy and returning feedback information (Weapon Control Organization, WCO). In each organization, a number of agents need to be defined to interact toward the achievement of a specific application goal.

C. Pattern-driven development

The HDA-based design pattern serves to fill this gap between design models and software implementation and guides designer to build a multi-agent system efficiently. Because the whole C4I system is too complex, only IPO (a single agent organization) is taken as an example to explain how to apply design pattern to facilitate development.

In the organization (figure 6), when coordinate agent receives requests from filter agent, it will assign the task to worker agent. It is our focus about how to choose the worker agent that "best" meets the attributes we care most about (e.g.

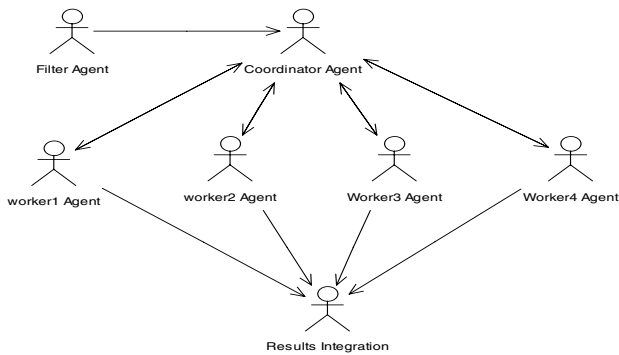


Fig. 6. The Information Process Organization

least expensive, highest quality). Market design pattern(as shown in figure 4) provides us a appropriate choice to implement the coordination mechanism of IPO. First coordinate agent requests worker agents to bid for the task. The coordinate agent selects the bidding whose cost is the least among the received bidding. Then the coordinate agent requests the worker agent with the successful bidding to perform the task. The application of market design pattern can ensure that the task can be executed the most effectively.

In a research project for C4I system development, the experimental results are satisfactory: on the one hand, the system meets the functional and non-functional demands(robust, reliable, scalable and so on); on the other hand, development efficiency have been improved greatly than before at a low cost.

IV. CONCLUSIONS

The definition of agent-specific methodologies is definitely one of the most explored topics in AOSE. Because of the defects of current AO methodologies (a variety in number, difficult to apply), it's difficult to turn AO software abstractions into practical tools for facing the complexity of constructing distributed systems. The paper presents a hierarchical development architecture (HDA) to enable developer to customize his own methodology to meet the demand of different domains. Through applying the HDA-based meta models and design patterns, developers can build applications from third party off-the-shelf solution components conveniently. In the case study of constructing C4I system, an agent-oriented new approach derived from HDA is applied and the experimental result is satisfactory.

In order to make full use of HDA, users have to understand and grasp more than one kinds of AO methodologies, which will become an obstacle in applying HDA. In future, we will carry on more research and studies on how to help users to overcome the difficulty. The idea was to create a relatively general purpose agent building toolkit that could be used by software engineers with only basic competence in agent technology to create functional MASs.

REFERENCES

[1] N.R.Jennings. An agent-based approach for building complex software system. *Commun, ACM*, vol.44, no.4, 35-41.

[2] M.Gervais, J.Gomez and G.Weiss. A survey on agent-oriented software engineering researches. *Methodologies and Software Engineering for Agent Systems*, Kluwer: NewYork(NY) (2004).

[3] F.Zambonelli, A.Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent System* (2004) 253-283.

[4] David kimny.et al. A Methodology and Modeling Technique for Systems of BDI Agents. In Proc of MAAMAW (1996).

[5] Khanh Hoa Dam and Michael Winikoff. Comparing Agent-Oriented Methodologies. In Proc of 5nd International Workshop on Agent Oriented Software Engineering (2003).

[6] O.Shehory, A.Sturm. Evaluation of modeling techniques for agent-based systems. In Proceedings of the 5th International Conference on Autonomous Agents, ACM Press:Montreal(2001).

[7] M.Saeki. Software Specification & Design Methods and Method Engineering. *International Journal of Software Engineering and Knowledge Engineering*, (1994).

[8] A.Kleppe, J.Warmer, and W.Bast. MDA Explained: The Model Driven Architecture: Practice and Promise. *Addison-Wesley, Object Technology Series, ISBN 032119442-X*, (2003).

[9] Henderson-Sellers, B., Debenham, J. Towards open methodological support for agent-oriented systems development. In Far, B., Rochefort, S., Moussavi, M., eds. Proceedings of the First International Conference on Agent-Based Technologies and Systems, University of Calgary, Canada, 14-24 (2003).

[10] T.Juan, L.Sterling, M.Martelli, V.Mascardi. Customizing AOSE Methodologies by Reusing AOSE Features. In Proc. of 2nd Int. Conference on Autonomous Agents and Multi-Agent Systems, Melbourne Australia (July, 2003).

[11] R.S.S.Guizzardi, V.Dignum, A.Perini, G.Wagner. Towards an Integrated Methodology to Develop KM Solutions with the Support of Agents. In Proc. of the International Conference on Integration of Knowledge Intensive Multi-Agent Systems, Waltham, Massachusetts (2005).

[12] Tom De Wolf and Tom Holvoet. Towards a Full Life-cycle Methodology for Engineering Decentralised Multi-Agent Systems. In Proc of the Fourth International Workshop on Agent-Oriented Methodologies (OOPSLA 2005), San Diego, USA (2005).

[13] T.Juan, L.Sterling, M.Martelli, V.Mascardi. Creating and Reusing AOSE Features. <http://www.cs.mu.oz.au/~tlj/CreatingAOSEFeatures.pdf>.

[14] T.Juan, S.Leon. The ROADMAP Meta-Model for Intelligent Adaptive Multi-Agent Systems in Open Environments. In AOSE, P53-68 (2003).

[15] Carole Bernon, Massimo Cossentino, Marie-Pierre Gleizes, etc. A study of some multi-agent meta-models. In Proc. of the Fifth International Workshop on Agent-Oriented Software Engineering, (2004).

[16] Thomas Juan. Leon Sterling. Michael Winikoff. Assembling Agent Oriented Software Engineering Methodologies from Features. In AOSE ,P198-209 (2002).

[17] T. Juan, A.Pearce and L.Sterling. ROADMAP: Extending the Gaia Methodology for Complex Open Systems. In Proc. of the First International Joint Conference on AAMAS, p3-10, Bologna, Italy (July 2002).

[18] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multi-agent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol*, 12(3):317-370 (2003).

[19] S.A.Deloach, M.F.Wood, C.H.Sparkman. Multiagent System Engineering. *Software Engineering and Knowledge Engineering*, 11(3):231-258 (2003).

[20] X.Tu, D.Terzopoulos. Artificial fished: physics, locomotion, perception, behavior. In Proc. of ACM Computer Graphics,Annual Conference Series, Proceedings of SIGGRAPH'94, pp.43-50 (1994).