

Developing MAS Solutions with Gaia and AUML

Luca Cernuzzi^{1,2}, Franco Zambonelli¹

1) DISMI – Università di Modena e Reggio Emilia, Italy

Via Allegri 13 – 42100 Reggio Emilia, Italy

2) DEI – Universidad Católica “Nuestra Señora de la Asunción”, Paraguay

Campus Universitario – C.C. 1683 Asunción, Paraguay, Tel: +595-21-334650, Fax: +595-21-310072

e-mail: lcernuzzi@uca.edu.py, franco.zambonelli@unimore.it

Abstract

A great number of Agent Oriented Software Engineering (AOSE) approaches and methodologies have been proposed in recent years, explicitly or implicitly, presenting different abstractions for the design and development of Multi-Agent Systems (MAS). This paper pays specific attention to the common abstractions used to model the complexity of MAS independently of the specific application. Moreover, it analyzes which of them are covered by Gaia integrated with AUML. Finally, as a case study, this paper presents an agent based solution modeled in Gaia with AUML and developed for the auditing process of the Central Bank in a developing country.

Keywords: Agent-based computing, Agent Modeling abstractions, Gaia, AUML.

1. INTRODUCTION

Agent-based computing represents an exciting new paradigm in the software engineering arena. Agents are being advocated as a next generation model for the engineering of complex, open and distributed systems. Yet, despite the increasing interest in these disciplines, some fundamental subjects, like the need for specific abstractions for modeling and developing MAS, have to be pointed out. In this sense, different authors have proposed different abstractions and different methodologies providing clear guidelines for the analysis, design, and development of MAS, and offering specific notations for each model [Ciancarini and Wooldridge, 2001], [Iglesias et al., 1999], [Jennings, 2000], [Jennings, 2001], [Cossentino and Zambonelli, 2004].

Agent-based computing includes a great variety of applications ranging from robot simulation to web distributed information systems. Because of this, it may be argued that these different types of applications may imply different abstractions. For example information searching or retrieval agents in internet may need mobility to move from one web site to another in order to accomplish their tasks, while intelligent agents claim for specific skills of reasoning.

However, independently of the application of any system in particular, our hypothesis is that some common abstractions can be identified for the agent-based computing paradigm.

Thus, a first question is: “is it possible to find convergence on some “must” abstractions for MAS independently of the specific application?”. If so, it would be possible to verify the adequacy of the proposed AOSE approaches and methodologies.

For this reason, we are interested in analyzing if Gaia integrated with AUML is a good candidate for MAS design and if it adequately captures these common abstractions. To do so, we also propose a case study centered on the design and development of an agent based solution, modeled in Gaia integrated with AUML, developed for the auditing process of the Central Bank of Paraguay.

The rest of the paper is structured as follows. Section 2 introduces a proposal for common agent-based computing abstractions; a brief summary of Gaia and AUML; and an analysis of how Gaia+AUML covers the proposed common abstractions. Section 3 presents the case study of the auditing system for the Central Bank of Paraguay. Section 4 discusses the proposal comparing it with related works. Finally, section 5 concludes and sketches future works.

2. COMMON ABSTRACTIONS IN GAI AND AUML

Different abstractions have been proposed to characterize MAS. Some of them are mainly dependent on the type of MAS that the designers take into consideration (e.g. MAS with mobility); while others are common abstractions that

are quite independent from the specific application. For generalization purposes, in this study we will mainly focus on this second group. Thus, among the common abstractions for MAS it is possible to identify:

- roles:
- activities with their corresponding goals
- agents:
- interaction with the environment:
- organizational aspects (structure and control):
- interaction between agents.

Currently, almost all traditional and emergent methodologies, as well as the engineering approaches proposed for agent-based systems design and construction, cover the mentioned abstractions. However, they do it in different ways, and some of them organize the process and the models in a more coherent framework than others. Some suggestions on this regard may arise from different interesting works on the evaluation of methodologies [Sturm and Shehory, 2003], [Cernuzzi and Rossi, 2002], [Hoa Dam and Winikoff, 2003], etc.

Considering this panorama, we are interested in analyzing how Gaia [Zambonelli et al., 2003], and its integration with AUML [Bauer et al., 2000; Odell et al. 2000], can be considered a good choice for designers. Therefore, in the next subsections we briefly introduce Gaia and AUML, and then analyze their adequacy in efficiently capturing the common abstractions for MAS.

2.1. Gaia in a Nutshell

Gaia [Zambonelli et al., 2003] focuses on the use of organizational abstractions to drive the analysis and design of MAS. Modeling both the macro (social) and the micro (agent internals) aspects of a MAS, Gaia devotes specific efforts to model the organizational structure and the organizational rules that govern the global behavior of the agents in the MAS organization. Gaia mainly covers the analysis and the architectural design phases.

The goal of the analysis phase in Gaia, which covers the requirements in term of functions and activities, is firstly to identify the loosely coupled sub-organizations that possibly compose the whole systems and then, to produce four basic abstract models: (i) the environmental model, to capture the characteristics of the MAS operational environment; (ii) a preliminary roles model, to capture the key task-oriented activities to be played in the MAS; (iii) a preliminary interactions model, to capture basic inter-dependencies between roles; and (iv) a set of organizational rules, expressing global constraints/directives that must underlie the MAS functioning.

The above analysis models are used as input to the architectural design phase. In particular, the architectural design phase is in charge of defining the most proper organizational structure for the MAS, i.e., the topology of interactions in the MAS and the control regime for the MAS which most effectively enables to fulfill the MAS goals. This definition of the organizational structure has to account for a variety of factors; including the need to somehow reflect the structure of the real-world organization in the MAS structure, the characteristics of the environment and of the patterns of access to it, the need of simplifying the enactment of the organizational rules, the need to fulfill any identified non-functional requirement, as well as the obvious need to keep the design as simple as possible. Once the most appropriate organizational structure is defined, the roles and interactions models identified in the analysis phase (which were preliminary, in that they were not situated in any actual organizational structure) can be finalized to account for all newly identified interactions and possibly for newly identified roles.

Past the architectural design phase, the detailed design involves identifying: (i) an agent model, i.e. the set of agent classes in the MAS, implementing the identified roles and the specific instances of these classes; and (ii) a services model, expressing services and interaction protocols to be provided within agent classes. The result of the design phase is assumed to be something that could be implemented in a technology-neutral way.

2.2. AUML IN A NUTSHELL

Several approaches address the problem of extending UML notation for agent-based systems. Some of these are: Agent UML [Odell et al., 2000; Bauer et al. 2000]; MESSAGE/UML (Methodology for Engineering Systems of Software AGENTS) [Caire et al., 2001]; AOR (Agent-Object Relationships) [Wagner, 2002]; PASSI [Cossentino and Sabatucci, 2004].

Among them, Agent UML (AUML) is particularly interesting because it builds on the acknowledged success of UML in supporting industrial-strength software engineering.

The core part of AUML is the Agents Interaction Protocol (AIP), specified by means of protocol diagrams. Protocol diagrams extensions to UML include agent roles, multithreaded lifelines, extended message semantics, parameterized

nested protocols, and protocol templates. AUML also proposes other extensions to UML in order to better capture more complete role specifications, packages with agent interfaces, deployment diagrams indicating mobility, emergence, etc. However, those notations are less rich than the ones proposed for AIP.

A three-layer representation of the AIP is defined as patterns representing both the message communication between agents and the corresponding constraints on the content of such messages.

1. The first layer represents the communication protocols. Protocols are modeled as whole entities (expressed by means of an enriched sequence diagram) that facilitates protocol reuse. Each protocol is treated as a package which aggregates modeling elements into a conceptual whole. Moreover, packages may be nested, so it is possible to define more reusable and easy to assemble protocols. AUML considers an AIP as a template, a parameterized model element whose parameters are defined at model time. These parameters are divided in three categories: roles, constraints, and communication acts.
2. The second layer represents interactions. The inter-agent interactions implementing the protocol are described using sequence diagrams, activity diagrams and statecharts that include some extensions to UML. Thus, these make possible to represent Agents (eventually specifying their Class) and their Roles. Also, instead of traditional OO messages the enhanced sequence diagrams specify the agent communication act. Moreover, an extension supports concurrent threads of interactions (considering: concurrence, inclusive or, and exclusive or). The activity diagram conveys operations and events that specify an explicit thread of control into an AIP. This process-centric view is particularly useful for complex interaction protocols that involve concurrent processing. Finally, the statecharts are useful for expressing constraints for AIP.
3. In the third layer activity diagrams and statecharts are used to specify the internal behavior of an Agent.

2.3. MAS ABSTRACTIONS AND GAIA + AUML

In other studies [Cernuzzi et al., 2004; Cernuzzi and Zambonelli, 2004] we have proposed to integrate the standard AUML notation into the Gaia process in order to make Gaia more expressive and easier to be accepted by software engineers. The proposal is mainly centered on adopting AIP notations from AUML instead of the Gaia ones. For simplicity and space reasons we call them Gaia+AUML. Thus, in this section we analyze the adequacy of Gaia+AUML for specifying all the proposed abstractions.

- *Roles*

Roles have been considered as the basis for agent design, since agents act by playing specific roles. AUML proposes some graphical notations, not always formal, for role modeling, as follow:

- i) a general class diagram which may be used to identify the tasks a role have to accomplish with. An agent may play different roles in the same sequence diagram and each node is a class of role;
- ii) collaboration diagrams, in which each communication act may be labeled with the related role;
- iii) activity diagrams, which may represent agent roles by associating them to each activity; moreover, changes in roles may be represented on activity diagrams using notes.

Additionally, the role specification in Gaia is more expressive, more formal and includes more relevant aspects than the AUML proposal. Specifically, Gaia includes the description of permissions and responsibilities into role modeling. Permissions specify the resources that are available for playing that role and, doing so, they create direct relationships between the role activities and the environment. The responsibilities of a role, on the other hand, define the role's actual functionality. There are two types of responsibilities: safety properties and liveness properties. Safety properties are properties that the agent acting in the role must always preserve and are expressed as predicates over the variables/resources in the permissions of the role. Liveness properties describe generalized behavioral patterns of the role and are represented by a regular expression over the sets of activities and protocols the role executes. In this sense, role modeling is quite rich compared to other similar models in alternative AOSE methodologies and we may conclude that Gaia+AUML adequately covers this abstraction.

- *Activities and Goals*

The activities and goals that characterize roles are related to the role specification. While both try to capture functional requirements, goals have been considered to be more general and stable than activities to specify the general behavior of the system to be. Both Gaia and AUML cover activities specification, but Gaia considers them as a part of the role diagrams in which both activities (tasks that an agent playing that role can accomplish by itself) and protocols (tasks that involve coordination with other agents playing specific roles) are captured. In AUML activity diagrams and statecharts are used to specify the internal behavior of agents and its activities, nevertheless, neither Gaia nor AUML

are focused on goal specification. Only a limited specification of the general goals of each role is provided in Gaia by means of informal descriptions, and for this reason, we may conclude that Gaia+AUMML may be improved to better cover this abstraction.

- *Agents*

Agent is the first class abstraction in the paradigm. As expected, both Gaia and AUMML support the definition of agents using some kind of agent class diagram and allow the specification of which roles a specific agent may play during its lifetime. Therefore, we may conclude that Gaia+AUMML adequately covers this abstraction.

- *Interaction with the Environment*

Normally, the context in which agents operate is composed not just of other agents, but also of human interactions, objects, and other kinds of resources. To capture those interactions, AUMML offers the use of sequence diagrams that consider the interaction with the environment objects. However, this may cause the designers to lose abstraction consistency when modeling objects, humans or other entities in addition to agents in the same model and with the same stereotypes. Gaia covers the interaction with the environment mainly capturing the resources that the agents in the organization need for accomplishing their functions. However, the notation is not entirely formal nor standardized, and thus, it is clear that Gaia+AUMML needs some improvement to better cover this abstraction.

- *Organizations and sub-organizations*

A lot of methodologies consider that an organization is just a set of roles; however, the same set of roles may assume very different types of organizations (i.e. hierarchy, network of peer, peers with a coordinator, etc.). Although Gaia explicitly considers structures of the overall organizations and sub-organizations, its notations are not well defined and do not offer specific guidelines for making design decisions. AUMML [Parunak and Odell 2002] presents some problems with organizational structures notation. In fact the use of class diagrams for the overall organization, subclasses (composition relationship) representing the sub-organizations, and inheritance for specific agents (that tend to consider an organization as a simple collection of roles) may be considered quite poor notations. Moreover, according to Huget, Odell, and Bauer [UML and Agents: Current Trends and Future Directions], even if groups are already considered through UML based diagrams, they are still immature. Therefore, we may conclude that Gaia+AUMML mainly covers this abstraction but needs more specific notations and guidelines.

- *Organizational rules*

The organizational structure may be insufficient to specify the general behavior of the MAS in terms of the interaction among the agents in the society, and for this reason, some methodologies have proposed to specifically capture the rules governing this organization. Gaia considers the organizational rules in a perspective that is coherent with the notion of responsibilities defined for roles, but referring to the organization as a whole. Accordingly, it is also possible to distinguish between safety and liveness organizational rules. Gaia adopts a formal defined notation based on regular expressions for rule specification, which is probably the more specific and formalized notation among the AOSE proposals. Therefore, we may conclude that Gaia+AUMML adequately covers this abstraction.

- *Interaction between agents (Protocols)*

Normally, agents operate in a context in which they need to cooperate, compete or just communicate (interact) with other agents. For this purpose AIP notation, proposed by AUMML and adopted in Gaia+AUMML, is now a *de facto* standard in the agent community. As we introduced in section 2.2, AUMML considers 3 levels to represent protocol diagrams (*communication protocol, interaction among agent, and internal agent processing*). Therefore, we may conclude that Gaia+AUMML efficiently covers this abstraction.

3. THE CENTRAL BANK AUDITING CASE STUDY

A multi-layer agent architecture for remote auditing in public institutions has been proposed to solve some deficiencies of the Central Bank of Paraguay (BCP – Banco Central del Paraguay), which is in charge of the auditing process of the financial-account management of credit firms. As the reader may observe in the proposed model of Figure 1, the BCP sends different agents to the financial institutions that the BCP needs to audit. In each financial institution the files are validated and modified to ensure correctness of information and format. Once the proper agent has completed its auditing activities, the obtained files are compressed and encrypted before being sent back to the BCP. Finally, the BCP sends an electronic receipt to the audited institution.

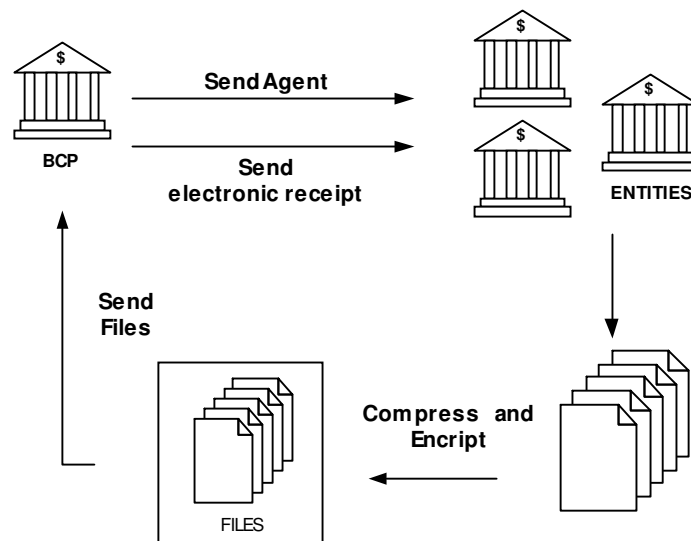


Figure 1. The proposed model

The experimental results achieved with the Auditing Agents BCP system, solve the problems identified in the current working model of the BCP.

3.1. THE BCP AUDITING SYSTEM IN GAIA + AUML

Possible sub-organizations

The auditing system includes four different processes: the correct format verification, the information validity verification, the verification of the relationships between files (also called referential integrity), and the verification of the sum of the debit for each client. Each verification process operates on different data files and includes multiple check activities, one for each of these four different files. Thus, it is natural to think in four different sub-organizations, one for each verification process.

Environmental Model

In this application the environmental model comprises the data bases of the audited banks (accessible if the agent has the corresponding permissions), a set of reports generated by the different agents, and the data base of the BCP.

Preliminary Roles Model

In this model designers capture the basic functional requirements of the application in terms of the roles that the different actors can play. Independently from the organizational structure, the main role is the Auditor which will be in charge of the general auditing process. Moreover, other roles are those related to each of the four verification processes previously identified. Other roles would may appear in the design phase. For space reasons we just present the Auditor role schema (see Figure 2).

Role Schema: Auditor
Description: Move to the audited bank and interact with the checker in order to identify the agent. If validated, it starts the auditing process coordinating the verification agents (format, validity, relation, and sum). Once it receives all the correct reports, it encrypts the DB and goes back to the BCP. If it is not validated, the auditing process is not allowed and the auditor has to go back to the BCP. Otherwise, each error detected during the auditing process is registered in a special file. The auditor stays in the bank until all the errors have been fixed.
Protocols y Activities: ReceiveConfiguration, SendKey, ReceiveConfirmation, RequestAuditFormat, RequestAuditValidity, RequestAuditRelation, RequestAuditSum, ReceiveFormatReports, ReceiveValidityReports, ReceiveRelationReports, ReceiveSumReport, <u>EncryptBD</u> , <u>GenerateFinalReport</u>
Permissions: reads: configuration // user configuration confirmation //boolean FormatReports (CA, CB, CC, CD) ValidityReports (CA, CB, CC, CD) RealtionReports (CA, CB, CC, CD) SumReport AuditedBD (CA, CB, CC, CD) generates: FinalReport //the auditing general report
Responsabilities: Liveness: <ul style="list-style-type: none"> AUDITOR = ReceiveConfiguration.SendKey.receiveConfirmation. FORMAT.VALIDITY.RELATION.SUM.(Error <u>EncryptBD</u>. <u>GenerateFinalReport</u>) FORMAT = RequestAuditFormat.ReceiveFormatReports VALIDITY = RequestAuditValidity.ReceiveValidityReports RELATION = RequestAuditRelation.ReceiveRelationReports SUM = RequestAuditSum.ReceiveSumReport Safety: <ul style="list-style-type: none"> $confirmation = true \Rightarrow informeFinal \neq null$ $ReportFCA=true \wedge ReportFCB=true \wedge ReportFCC=true \wedge ReportFCD=true \wedge ReportVCA=true \wedge ReportVCB=true \wedge ReportVCC=true \wedge ReportVCD=true \wedge ReportRCA-CC=true \wedge ReportRCB-CA=true \wedge ReportRCB-CC=true \wedge ReportRCC-CA=true \wedge ReportSum=true \Rightarrow encrypt(Audited\ BD)$ $confirmation = false \Rightarrow error$

Figure 2. The auditor role schema

Preliminary Interaction Model

As for the preliminary roles model, and although this model may experience changes once the organizational structure is defined, some preliminary interaction protocols could be identified. For this model we adopt the AIP notation from AUML, and a representative example of this is presented in Figure 3 (for space reasons all the formats, validities and relations verifications are grouped in one agent class for each type).

Neither Gaia's nor AUML's notations offer facilities to specify agent mobility. Therefore, to represent agent migration we proposed the adoption of dashed arrows and of dashed boxes (e.g. in figure 3 the agent Auditor moves from the BCP to other bank).

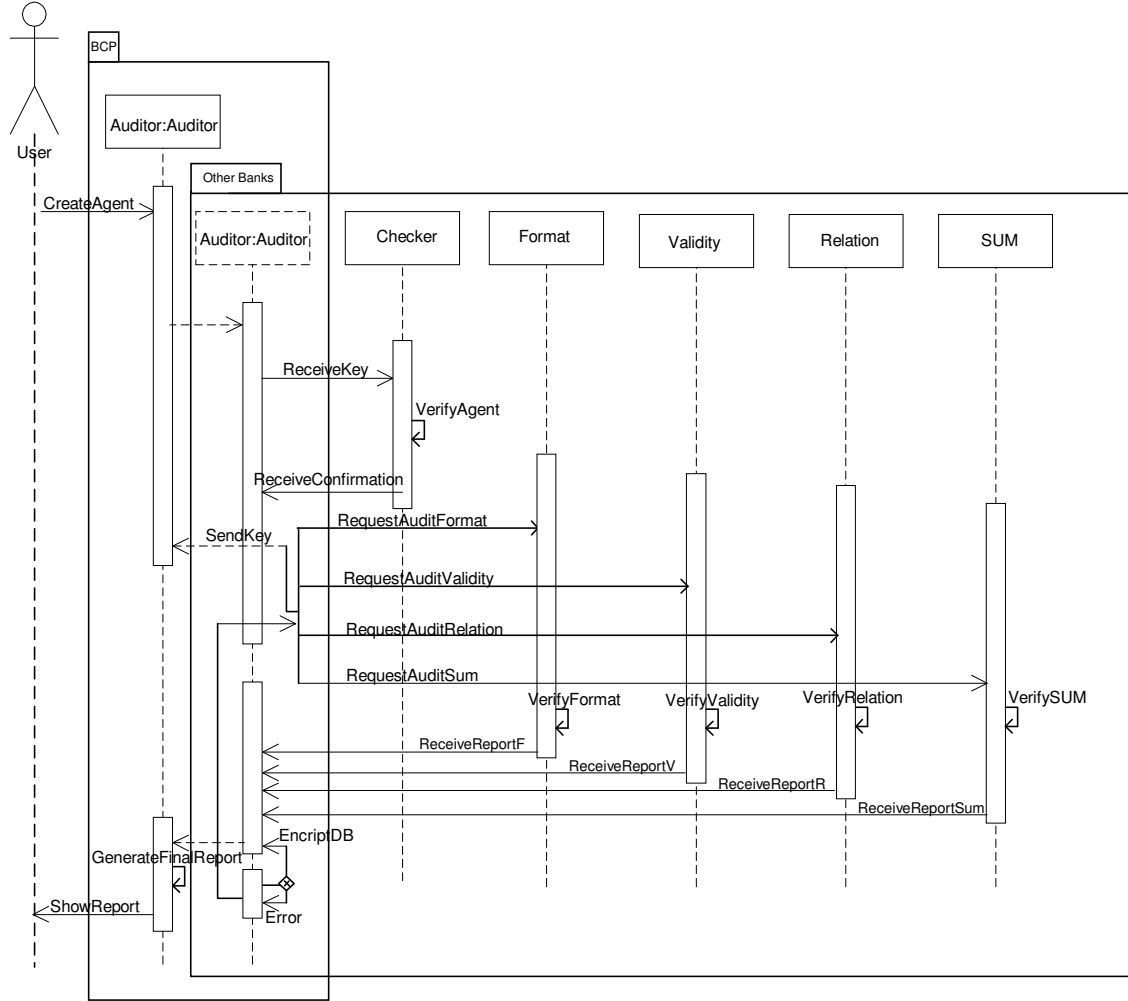


Figure 3. An AIP protocol model

Organizational Rules

The organizational rules are invariants that agents, playing the corresponding roles, have to observe for maintaining a correct behavior of the organization.

Different kind of rules may be defined for the BCP auditing system, but for space reasons we just present some examples of the complete set.

Some of these rules may also control the order of the activities. For example, rule 1 means that CreateAgent has to be executed before of ProgrammingAgent; while rule 2 states that ReceiveConfirmation precedes the warning for an ErrorKey or the request for a specific auditing process (in this case the format auditing).

Other rules may specify different constraints for the general behavior of the organization. For example, rule 4 means that if an Auditor has been validated all the corresponding reports (for Format, Validity, Relation and Sum) must exist.

1. CreateAgent \rightarrow ProgramminAgent
2. ReceiveConfirmation \rightarrow ErrorKey | RequestAuditFormat
3. Key \neq null
4. \forall Auditor, confirmation=true \rightarrow ReportFCA \neq null \wedge ReportFCB \neq null \wedge ReportFCC \neq null \wedge ReportFCD \neq null \wedge ReportVCA \neq null \wedge ReportVCB \neq null \wedge InformeVCC \neq null \wedge ReportVCD \neq null \wedge ReportRCA-CC \neq null \wedge ReportRCB-CA \neq null \wedge ReportRCB-CC \neq null \wedge ReportRCC-CA \neq null \wedge ReportSum \neq null
5. \forall Auditor, home=true \wedge FinalReport \neq null \rightarrow change BD BCP

Organizational Structure

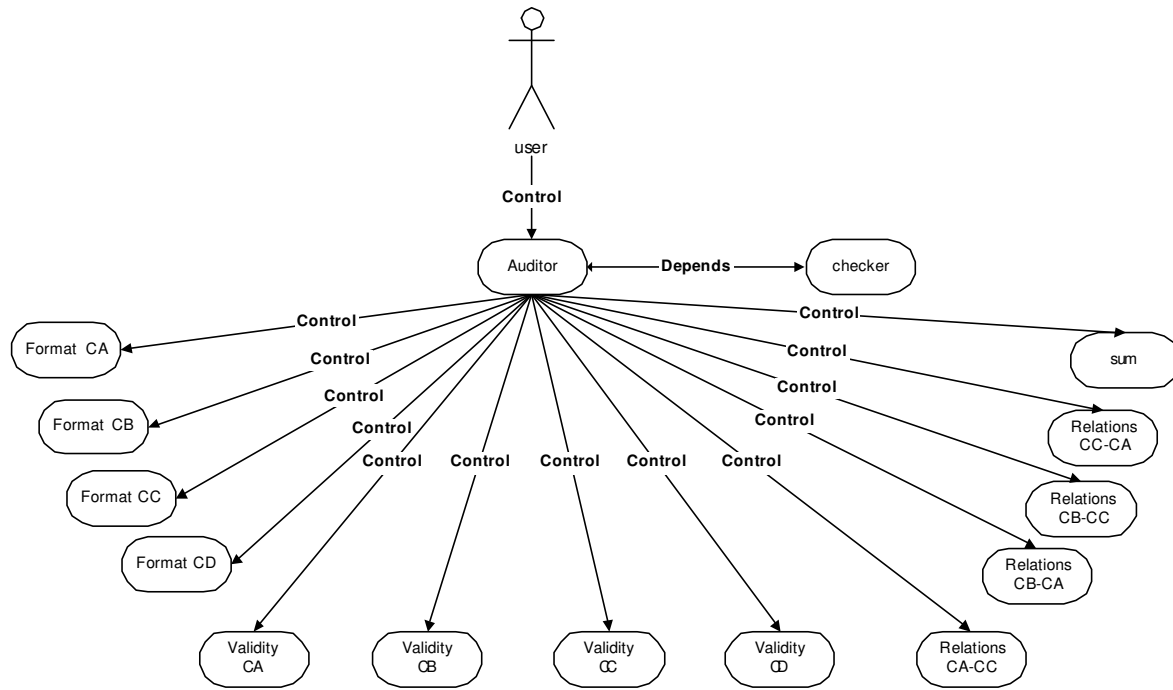


Figure 4. Organizational structure

Since the user has the control over the system, she/he is in charge of the creation of the auditor agent. This auditor agent in turn, depends on the checker agent that is in charge of auditor validation. The auditing process starts once the auditor creates all the agents in charge of the different verification processes for the audited bank. Those agents prepare and send a report according to the corresponding validation results. Since the auditor has created all the verification agents, it controls those agents and may eventually decide to eliminate them at any moment.

Detailed Design

The detailed design of Gaia considers the Agent Model, in which each role corresponds to a class, and the Service Model. For each auditing process, one Auditor and one Checker agents are instantiated while the other agents (Format, Validity, Relation, and Sum) may or may not be instantiated. If the Auditor agent is not recognized by the Checker these other agents are not instantiated, in all other cases, the verification agents are created when necessary until all errors have been corrected.

In the Service Model the designers identify the services associated to each class specifying inputs, outputs, pre and post-conditions. Inputs and outputs are derived from AIP (for the services associated to the agent interaction) and from the Environmental Model (for the services that operate on the resources). Pre and post-conditions derive from the safety properties of roles, from the organizational rules, from constraints on the availability of the resources, and from specific values of resources or data from other agents. For example the service “RequestAuditFormat” needs the ‘User Configuration’ as input; it has the pre-condition ‘Confirmation=True’, and the post-condition “FormatCA• NULL and FormatCB• NULL and FormatCC• NULL and FormatCD• NULL”.

Gaia and AUML do not cover the development and implementation stages. For those activities we adopt the Aglets Software Development Kit (ASDK).

4. DISCUSSION

During these last two years, our group has designed and developed different multi-agent systems using Gaia and Gaia+AUML. The Auditing system for the BCP is a representative example of them. Based on this experience, it is possible to analyze the advantages and limitations of Gaia+AUML in the analysis and design of MAS modeling common abstractions.

As already discussed in section 2.3, Gaia offers a general and coherent framework for the design and specification of MAS while Gaia+AUML adequately covers the roles, agents, organizations and sub-organizations, organizational rules, and interactions between agent abstractions. Moreover, the Gaia+AUML models and notations for roles and organizational rules are richer than those proposed by others methodologies. More specifically, both roles and organizational rules models adopt the permission and responsibilities notions that may formally express relevant specifications of the system's properties in a descriptive way. Also, the adoption of AUML notations for the specification of the interactions between agents is a trend in AOSE methodologies that has demonstrated to be useful in guiding the transformation from the design to the development of MAS.

Some limitations of Gaia+AUML are related with the notations adopted, in particular, the specification of the interaction with the environment and of the organizations and sub-organizations need to be improved. However, in the AOSE arena there are no standards that properly cover these aspects. Furthermore, Gaia+AUML still needs specific notations to capture specific aspects. As an example of this, we have proposed an extension of AUML to specify agents mobility in the BCP auditing system, however, this proposal is far to be accepted as a standard in the AUML community.

Finally, Gaia+AUML also suffers from some limitations with regards to the goals specifications. As advocated by different authors, goals may specify the general behavior and consequently the functional requirements of the system-to-be in a more stable way. This flaw is related to the way the methodology covers the requirement elicitation phase. However, as presented in section 2.1, Gaia framework does not include requirement elicitation activities and we consider that improvements in this aspect are a strongly needed.

Taking these limitations into consideration, it may be interesting to briefly analyze the approaches adopted by other methodologies.

An approach quite common among the methodologies influenced by the object orientation paradigm is to adopt the Use Cases diagrams to specify requirements. Some examples of such methodologies are PASSI [Cossentino and Sabatucci, 2004], Roadmap [Juan et al., 2002], and MaSE [DeLoach et al., 2001]. Nevertheless, Use Cases diagrams are not enough to capture all the functional and non-functional requirements of a MAS.

Another approach focuses on the goals specification. Some examples of methodologies that follow this approach are Tropos [Bresciani et al., 2001; Giunchiglia et al., 2002] and Prometheus [Padgham and Winikoff, 2002]. Among all these, the most interesting proposal is Tropos in which the requirements phase is strongly influenced by the i* modeling framework [Yu, 1995].

Therefore, we may conclude that a possible way to improve Gaia's general framework is to consider the requirement elicitation phase; perhaps by adopting similar solutions to the Tropos one, or even better, by exploring alternative ways that can be integrated into Gaia while still maintaining its coherence and harmony.

5. CONCLUSIONS AND FUTURE WORKS

Authors have proposed different abstractions and methodologies to provide clear guidelines for the analysis, design, and development of MAS; offering specific notations for each model. Also, since agent-based computing includes a great variety of applications, it may be argued that different types of applications may imply different abstractions. However, independently of the application of any system in particular, our hypothesis is that some common abstractions may be identified in all applications using this paradigm. Consequently, this paper focuses on the common abstractions used to model the complexity of MAS: roles, activities with their corresponding goals, agents, interaction with the environment, organizational aspects (structure and control), and interaction between agents.

The present work also analyzes which of these abstractions are covered by Gaia integrated with AUML. Moreover, the paper presents, as a representative example, an agent based solution modeled in Gaia with AUML and developed for the auditing process of the Central Bank of Paraguay.

We may conclude that Gaia offers a general and coherent framework for the design and specification of MAS and that, despite some limitations related with the notations adopted and the lack of requirements specification, Gaia+AUML adequately covers the roles, agents, organizations and sub-organizations, organizational rules, and interactions between agents abstractions.

Therefore, possible lines that may be explored in future works are concerned with the integration into Gaia of an explicit requirements elicitation model and some improvements to Gaia and AUML notations.

Acknowledgements

Special thanks to Professor José Bogarin, who co-directed the BCP project with the first author, and Anouk Twijnstra and Cynthia Villalba for their collaboration in the development of the Auditing Agents System for the BCP.

References

- Bauer, B., Müller, J., and Odell, J., 2000. Agent UML: A Formalism for Specifying Multiagent Software Systems. In: Ciancarini, P., and Wooldridge, M. (Eds.) *Agent-Oriented Software Engineering - Proceedings of the First International Workshop (AOSE-2000)*. Springer-Verlag, Berlin (Germany) , pp. 91-103
- Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J., 2001. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In: *Proceedings of the 5th International Conference on Autonomous Agents*. ACM Press, Montreal (Canada), pp. 648-655
- Caire, G., Chainho, P., Evans, R., Garijo, F., Gómez Sanz, J., Kearney, P., Leal, F., Massonet, P., Pavón, J., 2001. Agent Oriented Analysis Using MESSAGE/UML. *Proceedings of Agent-Oriented Software Engineering – AOSE 01*, May 2001, Montreal (Canada), pp. 101-107
- Cernuzzi, L. and Rossi, G., 2002. On The Evaluation Of Agent Oriented Methodologies. *Proceedings of the OOPSLA 02 - Workshop on Agent-Oriented Methodologies*, November 2002, Seattle (USA), pp. 21-30
- Cernuzzi, L., Juan, T., Sterling, L., and Zambonelli, F., 2004. The Gaia Methodology. (Chapter book) in *Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering handbook*. F. Bergenti, M.-P. Gleizes and F. Zambonelli Editors, (pp. 69-88), Kluwer Publishing, 1-4020-8057-3
- Cernuzzi, L., and Zambonelli, F., 2004. Experiencing AUML in the Gaia Methodology. In: *Proceedings of 6th International Conference on Enterprise Information Systems - ICEIS 2004* (Vol. 3, pp. 283-288), Porto, Portugal, April 2004
- Ciancarini, P. and Wooldridge, M., 2001. Agent-Oriented Software Engineering. *Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering*, Springer Verlag, LNCS, Vol. 1957, pp. 1-24
- Cossentino, M. and Sabatucci, L., 2004. Agent System Implementation in Agent-Based Manufacturing and Control Systems: New Agile Manufacturing Solutions for Achieving Peak Performance. Paolucci M. and Sacile R. editors. CRC Press, April 2004
- Cossentino, M. and Zambonelli, F., 2004. Multiagent Systems Development from the Autonomy Perspective, *Computational Autonomy*. LNCS Series No. 2969, Elsevier Ed.
- DeLoach, S., Wood, M. and Sparkman, C., 2001. Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering*, vol. 11, No. 3, pp. 231-258
- Giunchiglia, F., Mylopoulos, J. and Perini A., 2002. The Tropos Software Development Methodology: Processes, Models and Diagrams. *Proceedings of Agent-Oriented Software Engineering (AOSE-2002)*, July 2002, Bologna (Italy), pp 63-74
- Hoa Dam, K., and Winikoff, M., 2003. Comparing Agent-Oriented Methodologies. *Proceedings of Agent Oriented Information Systems-AOIS'03*, July 2003, Melbourne (Australia), pp. 78 - 93
- Iglesias, C., Garijo, M. and González, J.C., 1999. A survey of Agent-Oriented Methodologies. In: Muller, J.P., Singh, M., and Roa, A.S. (Eds.), *Intelligent Agent V*, Proceeding of ATAL-98, Springer, LNCS 1555, pp. 317-330
- Jennings, N., 2000. On agent-based Software Engineering, *Artificial Intelligence* 117, pp. 277-296.
- Jennings, N. R., 2001, An Agent-Based Approach for Building Complex Software System, *Communications of the ACM*, Vol. 44, No. 4, pp. 35-41.

- Juan, T., Pearce, A. and Sterling, L., 2002. ROADMAP: Extending the Gaia Methodology for Complex Open Systems. Proceeding of the First International Conference on Autonomous Agents and Multi-Agent Systems - AAMAS '02, July 15-19, 2002, Bologna (Italy), pp. 3-10
- Odell, J., Parunak, H. v. D., and Bauer, B., 2000. Extending UML for Agents. Proceedings of Workshop on Agent Oriented Information Systems – AOIS 2000, Austin, USA
- Padgham, L. and Winikoff, M., 2002. Prometheus: A Methodology for Developing Intelligent Agents. Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems - AAMAS '02, Third International Workshop on Agent-Oriented Software Engineering AOSE-2002, July 15, 2002, Bologna (Italy), pp. 135-146
- Sturm, A., Shehory, O., 2003. A Framework for Evaluating Agent-Oriented Methodologies. Proceedings of Workshop on Agent-Oriented Information Systems – AOIS 2003, 5th International Bi-Conference, July 14, Melbourne, (Australia), October 13, 2003 and Chicago, IL (USA), 2003, LNCS 3030, pp. 94-109
- Wagner, G., 2003. The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior. *Information Systems*, Vol. 28, No. 5, July, 2003, Elsevier, pp. 475-504
- Yu, E., 1995. Modelling Strategic Relationships for Process Reengineering. PhD thesis, University of Toronto, Department of Computer Science
- Zambonelli, F., Wooldridge, M. and Jennings, N. R., 2003. Developing Multiagent Systems: The Gaia Methodology. *ACM Transaction on Software Engineering and Methodology*, vol. 12, No. 3, pp. 417-470