Methodologies for Designing Agent Societies

Giacomo Cabri, Letizia Leonardi, Mariachiara Puviani
Dipartimento di Ingegneria dell'Informazione – Università di Modena e Reggio Emilia
Via Vignolese, 905 – 41100 Modena – ITALY
E-mail: {giacomo.cabri, letizia.leonardi, mariachiara.puviani}@unimore.it

Abstract

Today's complex systems can be addressed by exploiting software agents, and in particular sets of interacting agents. Agent societies represent a powerful means to design organization-oriented agent-based systems, in particular those where agents belonging to different users meet and interact. To effectively exploit them, developers must be supported by appropriate methodologies. In this paper we present a survey of agent-based methodologies, evaluated under some criteria that, in our opinion, are useful for developing societies. Our aim is to help developers in understanding the society-related features and choosing the most appropriate to their needs.

1. Introduction

Software agents represent a useful paradigm in the development of complex distributed systems. The feature of *sociality* of agents enables building systems composed of several agents, where they interact to achieve a common goal (cooperative agents) or to exploit each other features to achieve self-interested goals (competitive agents).

Nowadays, developers interested in an architecture based on services have to work not on a single agent, but on a group of agent. So "organizations" are a set of entities, ruled by mechanisms of social order, created by more or less autonomous actors, to achieve common or single goals [10]. Starting from this definition, Multi Agent Systems (MAS) are the natural way to design organizational systems, because of their ability to reorganize themselves dynamically as their problems and constituted agents change [7], and because of their proactivity and autonomy, which, together with reactivity and the already-mentioned sociality, are the main characteristics of agents.

On the one hand, in complex systems MAS are usually considered from an individualistic perspective: an aggregation of agents that simply interact with each other; and most of MAS architectures are closed, meaning that participant agents must be designed in a given architecture. On the other hand, structural organization determines important autonomous activities that must be explicitly organized into autonomous entities and relationships in the conceptual model of the *agent society* [10]. Agent societies can well model organizations because they provide mechanisms to allow them to foster their capabilities, negotiate their terms, exchange vast amount of information, and synchronize processes and workflow at a high-level of abstraction.

In this paper, we consider MAS that constitute agent societies. As in [10], we assume that the agents participating in a society are designed and developed also using different methodologies, outside the scope and design of the society itself. Therefore the society should deal with organizational and normative elements, interactions, desires and beliefs of participating agents, but has not to directly depend on them to be developed. The exploitation of an appropriate methodology can help developers in defining and building agent societies that can enrol different agents.

The aim of this paper is to present a survey of agent methodologies, performed on the base of proposed criteria. These are used to evaluate their orientation towards the support for developing agent societies. In our research, we have considered several interesting approaches to design agent societies, but in this paper we report only the most interesting ones, due to space limitation.

2. Agent Societies

The most important service that a society provides is to rule the interactions between members; so we can define different kinds of societies from different coordination models. According to [10] and [11], we can say that coordination of agent societies follows three different models: *market*, *network* and *hierarchy*. But the main difference is between *open* and *closed* societies. Open societies have no enrol restrictions and they support openness and flexibility (the degree of restriction in agent's behaviour); on the contrary, in closed societies it is impossible for external agents to join the society, and they support stability and trustfulness.

0-7695-3109-1/08 $25.00 © 2008 IEEE
DOI 10.1109/CISIS.2008.65

In general, an agent society should support *openness*, *flexibility*, *stability* and *trustfulness*. Unfortunately, neither open nor closed societies support all these features; so, according to [8], we consider semi-open and semi-closed societies. Semi-open societies seem better since only slightly limit the openness and have a much larger potential for providing stability and trustfulness, while semi-closed societies provide the same degree of openness but are less flexible (see [8] for further details).

With regard to complex distributed systems, we remark that openness is a key feature that cannot be discarded in their development.

3. Society Requirements

If participating agents are assumed to be designed outside the society itself, modelling an agent society cannot be based on assumptions about the internal behaviour, beliefs or actions of individual agents. Then, considering [10] and [11], in the following we propose some important requirements that must be taken into account when evaluating agent methodologies under the agent societies' point of view:

- they have to support and lead the analysis of the organizational structure of the domain, in order to determine society norms and facilitation roles;
- they must include mechanisms for the construction and control of the organisational and normative elements of a society (such as roles, rules, norms and goals), the environment and the society interactions;
- they should provide building directives concerning the communication capability and the ability to conform to the expected role behaviour of agents participating in the society;
- they should be provided by tools to implement societies and verify them.

From the previous considerations, the main aspects that a methodology has to take into consideration, besides agents, are:

- The structure that is determined by **roles**. Roles allow the members of the society to coexist in a shared environment, and to pursue their respective goals. Each agent plays a role in the society, which makes it acts and interacts in the society itself.
- **Rules**, **constraints** and **norms** that describe the desirable behaviour of agents in the society. Every agent needs rules to follow, to join the society, constraints and norms are needed to communicate, to achieve goals, and to build the society "hierarchy".
- **Goals**, which are the overall objectives of the society, desired by the society "owners";
- The **communication** framework that describes the interactions between agents. It includes the description of the society ontology (vocabulary understood within the society), the communication language and the representation language for domain content.

Considering these aspects in an appropriate way allows to correctly design an agent society, so their presences in methodologies are taken as criteria to be satisfied for building a society.

4. Methodologies

In the following, we present some known methodologies (in alphabetic order) evaluated under the aspects sketched in the previous section.

ADELFE. ADELFE defines a methodology to develop applications in which self-organization makes the solution emerge from the interaction of its parts, and it guarantees that the software is developed according to the AMAS (Adaptative Multi-Agent System) theory [3], using a notion based on UML/AUML.

In ADELFE the **role** notion is useless because designers have only to focus on the ability of an agent to detect and solve cooperation failures. If a designer gives roles to agents, by describing tasks or protocols, they will establish a fixed organization for these agents, and this is not welcome for building open societies. So in ADELFE roles are considered much strictly than roles in agent societies.

Instead, **constraints** are on average considered. They are first analyzed in the *Characterization of Environment* step. Then, in the *Design* phase, during the *Agent Design* activity, **rules** are identified, in terms of how to detect and to remove Non Cooperative Situations [15]. In fact, rules allow agents to have a cooperative attitude. Moreover, to enable the developer to deal with the specific components of the ADELFE methodology, nine stereotypes are defined, and each of these is characterized by rules.

The notion of **goal** does not directly appear in this methodology. The goal an agent has to achieve is modelled by its skills, aptitudes, and representations.

In the *Final Requirements* phase, to characterize the environment, **interactions** between agents and the system are expressed. These are very important mainly for the society relationships. Then in the *Design* phase, relationships between agents are addressed: designers have to define protocol diagrams that explain how the agents are going to interact and for each agent its **interaction language**. In the *Study of Interaction Language* activity, a set of protocol diagrams representing the different interaction languages used by

the agents are developed. AUML notation is used but some specific functionalities are added to diagrams, to fit AMAS theory requirements. Interactions, in this methodology are therefore very well analyzed.

GAIA. Gaia was the first methodology proposed to guide the process of developing a MAS, from analysis to design, but in its first version suffers from several limitations: it is suitable only for designing closed MAS, and the notions it uses are not suitable for dealing with the complexity of real-word [4]. To overcome only the first limitation, which is very important for agent societies, we analyze an extension of Gaia, Gaia v.2 [18].

Roles in Gaia are strongly considered. In the *Analysis* phase, roles in the system are identified, and their interactions are modelled. Here a role is an abstract construct used to conceptualise the system, but does not necessarily have any direct realisation within the system. All roles are considered as atomic constructs, and cannot be defined in terms of other roles. Role schemas are used as a semi-formal description of an agent's behaviour; the collection of all the role schemas describes the complete role model. Gaia v.2, differently from Gaia, leaves the role model incomplete, assuming that only an accurate identification of the organizational structure – done in the Design phase – will enable exact understanding of which role will interact with which other. In the *Design* phase, an *Agent Model* is created, where each role is assigned to an agent type.

In the *Analysis* phase, a set of organizational **rules** for each of the sub-organizations that compose the overall system is identified. Rules, adopting some specific organizational structures (for example constraints specifying that two roles have to be played by the same agent or that the same agent cannot play two specific roles concurrently), have to specify **constraints** that the organization has to observe [19]. They can be liveness rules or safety rules. These rules are global (concerned with all roles and protocols) or just concern the relations of some roles and/or protocols. They govern the organization in its global behaviour, explicitly defining when, where and under which conditions a new agent can participate in the society. In the *Design* phase, a *Service Model* is created: for each service that may be performed by an agent, it is necessary to document its properties, in particular it must be identified the inputs, outputs, pre-conditions and post-conditions. The last two ones represent constraints on the execution and completion of services. So this criterion is very well satisfied here.

The identification of the **goals** of the organizations (society's goals) that constitute the overall system and their expected global behaviour is done in the *Analysis* phase, in the *Environmental Model*. This step is very important to identify useful decomposition of the global organization in sub-organizations, but it is not so deeply analysed.

Interactions instead, are well analyzed because in the *Analysis* phase of Gaia v.2, in addition to the *Role Model*, there is the definition of a preliminary *Interaction Model*, which captures the dependences and relationships between the various roles in the MAS organization. The model defines one protocol for each type of inter-role interactions; the interaction model will be completed in the *Design* phase. To supply at the **language** limitation, a connection between GAIA and AUML has been introduced [4].

MaSE. Multiagent System Engineering is a full-lifecycle methodology for analyzing, designing and developing heterogeneous MAS; it is independent of any particular agent architecture, programming language, or communication framework.

In MaSE, **roles** are very well treated, especially in the final step of the *Analysis* Phase, called *Refining Roles*. Its purpose is to transform the *Goal Hierarchy Diagram* (GHD) and the *Sequence Diagram* into roles and their associated tasks, which are the form more suitable to design MAS. In this phase a *Role Model* is constructed: it describes the roles in the system and also depicts the goals that those roles are responsible for, the tasks each role performs to achieve its goals and the communication path between the roles.

Tasks can be seen as **rules**, and are generally derived from the goal of which they are responsible, but they are not so well treated in this methodology.

The first step of the *Analysis* phase is *Capturing Goals*, which aims at transforming an initial system specification into a set of structured system **goals**. There are two sub-steps: *Identifying Goals* and *Structuring Goals*. First, goals must be identified from the initial system context, capturing the essence of an initial set of requirements. Next, the goals are analyzed and put into hierarchical form: a GHD, a directed, acyclic graph where the nodes represent goals and the arcs define a sub-goal relationship [9]. So the goal criterion is very well satisfied.

Otherwise **interactions** are on average considered. In the *Design* phase there is a *Constructing Conversation* step where, using two *Communication Class Diagrams*, one for each the initiator and the responder, a conversation is defined as a coordinator protocol between two agents. *Communication Class Diagrams* use states and transitions to define inter-agent interactions.

PASSI. The Process for Agent Societies Specification and Implementation (PASSI) is a step-by-step requirement-to-code methodology for

designing and developing Multi-agent societies, using the UML notation. PASSI uses standards whenever possible. It considers two different aspects of agents: during the initial step of design, they are seen as autonomous entities pursuing an objective through autonomous decisions, actions and social relationships; then they are considered as part of a system.

An agent can play several functional **roles** during interactions with other agents to achieve its goals. A role is a social concept: a collection of tasks performed by the agent in following a sub-goal or offering some services to the other members of the society. In the *System Requirements Model* there is the *Role Identification* phase, where a series of sequence diagrams, exploring the responsibilities of each agent through role-specification scenarios, are created. In the *Agent Society Model* there is the *Role Description* step, where *Class Diagrams* are used to show the role played by agents, the tasks involved, communication capabilities, and inter-agent dependencies. Roles are very important for this methodology and so they are very well treated.

Otherwise **constraints** are not so deeply treated. In the *Ontology Description* step of the *Agent Society Model*, they are used to describe the knowledge of each agent, with the help of OCL (Object Constraints Language). Society rules (organizational rules) are then introduced in the *Role Description* phase, into the *Role Description Diagram*, and can be expressed in plain text or in OCL.

In PASSI, **goals** are not so deeply treated.

Instead **interactions** are very well treated; the *Agent Society Model* is a model of the social interactions and dependencies among the agents. It is made of three steps: *Ontology Description*, where class diagrams and OCL constraints are used to describe the knowledge ascribed to individual agents and their communications; *Role Description*, already mentioned; and *Protocol Description*, where sequence diagrams are used to specify the grammar of each pragmatic communication protocol, in terms of speech-act performatives [5].

Prometheus. Prometheus is a methodology for developing agent-oriented systems, and it aims at covering all the stages of software development.

Here **roles** are defined as functionalities, analyzed during the *System Specification* phase. They have to be kept as narrow as possible, dealing with a single aspect or sub-goal of the system, in fact, if functionalities are too broad they are likely to be less adequately specified leading to potential misunderstanding. In defining a functionality, it is important to define information required and produced by it; each functionality should be linked to some system goals [14]. Roles definition is also used in the *Architectural Design* phase, in constructing a *Data Coupling Diagram* that consists of the functionalities and all identified data, but they are not so deeply treated.

Likewise constraints are not well treated, even if from the *Data Coupling Diagrams*, we can extract **constraints** that can be useful to build agents.

System's **goals** and functionalities are determined in the *System Specification* phase, and they are strongly considered. The determination of goals is an iterative process: identify and refine system goals, group goals into functionalities, prepare functionality descriptors, define use case scenarios (that help to identify missing goals), check that all goals are covered by scenarios. An initial set of goals is identified from the initial requirements; they are refined and elaborated into hierarchy of goals by asking how goals will be achieved, and why goals are being achieved [17].

In the same way also **interactions** are well treated. In the *Architectural Design* phase, using *Interaction Diagrams* (developed from scenarios) and *Interaction Protocols* (developed from interaction diagrams), the interactions between agents are identified and described. Before this step, information about agent interactions are extracted from the functionality descriptors and each agent type is linked with the other agent types it interacts with. The *Describing the Interaction between Agents* sub-phase focuses on the system's dynamic behaviour, specifying the interaction between agents. *Interaction Diagrams* borrowed from UML sequence diagrams are an initial representation of agent interactions, then they are specified with interaction protocols that are the final design artefact.

SODA. SODA (Societies in Open and Distributed Agent spaces) is a methodology for the analysis and design of Internet-based applications as multi-agent systems. The aim of SODA is to define a coherent conceptual framework and a comprehensive software engineering procedure that accounts for the analysis and design of individual agents, agent societies, and agent environments. It is not concerned with intra-agent issues: designing a MAS with SODA leads to define agents in terms of their required observable behaviour and roles in the multi-agent system [13].

In the *Architectural Domain*, the solution domain is analyzed and the system is modelled also in terms of **roles**, here very well analyzed. Roles, defined in a *Role Table*, are an abstraction responsible for the achievement of one or more tasks. In the *Analysis* phase the role model is build, each individual task is associated to a role, while each social task is associated to a group. However, at a finer-grain level of detail, also each group is associated to a set of roles. In the *Design* phase, in the *Agent Model*, agent classes are

defined as a set of roles, both individual and social ones. As a result, an agent class is first characterised by the tasks, the set of the permissions, and the interaction protocols associated to its roles.

Interactions are defined as "**rules**" with the aim to enable and bound both the behaviour of the abstract entities and the space of interactions, and they are good considered. Constraints are relationships among abstract entities and are modelled in the *Analysis* phase. In the *Design* phase, the first point in the design of agent societies is the choice of the fittest coordination model – that is, the one providing the abstractions that are expressive enough to model the society interaction rules. Thus, a society is designed around coordination media embodying the interaction rules of its groups in terms of coordination rules.

During the *Analysis* phase, the application domain is studied and modelled, and the fundamental application **goals** and targets are pointed out and modelled in terms of (individual and social) tasks to be achieved. This phase should take into account and model the required and desired features of the agent application environment, by modelling it in terms of the required resources and the services made available to agents. So goals are very well treated.

Like goals, are interactions; since agents are basically interactive entities, which depend on other agents and available resources to pursue their tasks, during the *Analysis* phase an *Interaction Protocol* is modelled in terms of the information required and provided by agents and resources. The **interactions** involve roles, groups and resources, and are modelled in terms of interaction protocols, expressed as information required and provided by roles and resources, and interaction rules, governing interactions within groups.

Tropos. Tropos methodology emphasizes Early Requirement Analysis and adopts Eric Yu's *i** model [16], which offers the definitions of actors, goals, roles and actor dependencies as primitive concepts for modelling the applications. In Tropos, organizational architectural styles for cooperative, dynamic and distributed applications are defined to guide the design of the system architecture.

A **role** is defined as an abstract characterization of the behaviour of a social actor within some specialized context or domain of endeavour. So roles in Tropos methodology are at average treated.

In the *Development* process, **constraints** are added to specifications, to capture more of the semantics of the subject domain, but they are not more considered.

Tropos is strongly focused on **goals**, which represent the intentions of stakeholders. They are divided into two different types: *hard* goals and *soft* goals. Hard goals eventually lead to functional requirements while soft goals, whose satisfaction conditions cannot be precisely defined, relate to non-functional requirements. In the *Early Requirement* phase, a *Goal Diagram* is built, and it shows the analysis of goals and plans with regard to the specific actor who has the responsibility for achieving them. There are also several techniques (means-end analysis, AND/OR decomposition, contribution analysis) that help the analysts to structure the system's goals, identify soft goals, plans and resources providing the means for accomplishing a goal and capture goals that promote or interfere with the fulfilment of other goals. In the *Late Requirements* phase, goals founded in the previous phase are decomposed into sub-goals [6].

In the *Early Requirement Analysis* phase, with the *i** framework, a *Strategic Dependency Model* is defined; it describes the network of relations among actors. The *Detail Design* phase includes the specification of agent **communication** and agent behaviour. To support this task, existing agent communication languages can be adopted, such as FIPA-ACL, and extensions to UML, such as the Agent Unified Modelling Language (AUML). The analysis in the Tropos framework is based on Formal Tropos (FT), a specification language that offers all the standard mentalistic notions of Tropos and supplements them with a rich temporal specification language inspired by KAOS [12]. Therefore, interactions are very important for this methodology, and very well treated.

Table 1: Methodologies' summary

	ADELFE	Gaia	MaSE	PASSI	Prometheus	SODA	Tropos
Roles	-	H	H	H	L	H	M
Rules, norms and constrains	M	H	L	M	L	M	L
Goals	-	L	H	-	H	H	H
Interactions	H	H	M	H	H	H	H

5. Discussion

In Table 1 we have summarized how the different methodologies satisfy the selected criteria. Each criterion is evaluated in a scale (High, Medium, or Low), where '–' means that it is not consider in that methodology.

Another important aspect to consider is the availability of tool support, because this enables the effective exploitation of a methodology during the development phases. In our study, we have found out

that almost all methodologies are supported by one or more tools, apart from SODA.

Norms are not directly taken into account in the considered methodologies, because they are not methodologies specific for building societies. But we can remark that sometimes constraints are often mapped on norms because the concept of constraint subsumes the concept of norm.

Concerning the aspects evaluated in this paper, we can see that all the considered methodologies well address interaction protocols. This is very important because it is the base for communications between agents and between agents and the system. Instead, we can notice that less methodologies support the identification of goals and even less ones enable the specification of rules. This can be explained by the fact that methodologies were mainly thought for MAS, where agents have individual goals, and although they need to interact, such interactions are not subject to "environmental" rules, as happens in the societies.

Our study has outlined that roles are the most important aspect for designing agent societies, so we can suggest developers to use this as first criterion. Of course, it must be mediated with the other requirements, but we feel that the identification of the system goal(s) could be the second criterion.

6. Conclusions and future work

Agent societies emerge as useful in modelling complex software systems, and an appropriate support is needed to develop them. In this paper, we have surveyed some agent methodologies to design agent societies. Our evaluation takes into consideration some aspects that we thought important for agent societies: roles, rules, goals and interactions. They are not always considered; in particular, we found that the most methodologies address the interaction issue, but less support the definition of system goals, and only one has a strong support for the specification of rules.

Our future work concerns two directions. First, we are going to evaluate the connection between existing methodologies and existing infrastructures: while both support the development of agent systems, there is a significant gap between the two worlds. We will evaluate the common parts between infrastructures and then we will analyse how they are treated in the previous methodology. Then, starting from these considerations and from the results of this paper, we are going to extrapolate methodologies' important fragments and propose different "threads of fragments" connected to possible scenarios, such as societies or services [2]. In this way we aim at building a composed (of different fragments) methodology, that can be useful and complete, and which can be entirely supported by infrastructure.

Acknowledgments. Work supported by the Italian MiUR in the frame of the PRIN project MEnSA - Agent oriented methodologies: engineering of interactions and relationship with the infrastructures.

References

[1] F. Bergenti, M.P. Gleizes, F. Zambonelli, "Methodologies and software engineering for agent systems", Kluwer Academic Publishers, 2004.

[2] G. Cabri, L. Leonardi, M. Puviani, "Service-Oriented Agent", *Proc. of WETICE 2007*, Paris, F, June 2007.

[3] D. Capera, J. P. Georgé, M.P. Gleizes, P. Glize, "The AMAS theory for complex problem solving based on self-organizing cooperative agents", *Proc. of WETICE 2003*, Linz, A, June 2003.

[4] L. Cernuzzi, T. Juan, L. Sterling, F. Zambonelli, "The Gaia Methidology", in [1].

[5] M. Cossentino, "From Requirements to Code with the PASSI Methodology", *Agent-Oriented Methodologies*, 2005.

[6] K.H. Dam, M. Winikoff, "Comparing Agent-Oriented Methodologies", *Proc. of the 5th International Bi-Conference Workshop on Agent-Oriented Information Systems*, Melbourne, 2003.

[7] M. Dastani, V. Dignum, F. Dignum, "Role assignment in open agent societies", *Proc. of AAMAS*, Melbourne, 2003.

[8] P. Davidsson, "Categories of artificial societies", *Engineering Societies in the Agents World II*, LNAI 2203, Springer-Verlag, 2001.

[9] S. DeLoach, "The MaSe Methodology", in [1].

[10] V. Dignum, F. Dignum, "Modelling agent societies: Co-ordination frameworks and institutions", *Proc. of EPIA*, 2001.

[11] V. Dignum, H. Weigand, "Towards an Organization-Orientes Methodology for Agent Societies", *Intelligent Agent Software Engineering*, Idea publisher, 2003.

[12] P. Giorgini, M. Kolp, J. Mylopoulos, M. Pistore, "The Tropos Methodology: An Overview", in [1].

[13] A. Omicini, "SODA: Societies and Infrastructures in the Analysis and Design of Agent based Systems", *Agent-Oriented Software Engineering*, LNCS 1957, Springer-Verlag, 2001.

[14] L. Padgham, M. Winikoff, "Prometheus: A pragmatic methodology for engineering intelligent agents", *Proc. of OOPSLA*, 2002.

[15] G. Picard, M.P. Gleizes, "The Adelfe Methodology", in [1].

[16] E. Yu, L. Liu, "Modelling Trust in the i* Strategic Actors Framework", *Proc. of the 3rd Workshop on Deception, Fraud and Trust in Agent Societies*, Spain, 2000.

[17] M. Winikoff, L. Padgham, "The Prometheus methodology", in [1].

[18] F. Zambonelli, N. Jennings, M. Wooldridge, "Developing multiagent systems: The Gaia methodology", *ACM Trans. Software Eng. Meth.*, vol. 12, No. 3, 2003.

[19] F. Zambonelli, N. R. Jennings, M. Wooldrige, "Multi-Agent Systems as Computational Organization: The Gaia Methodology", *Agent-Oriented Methodologies*, 2005.