SONIA: A Methodology for Natural Agent Development

Fernando Alonso, Sonia Frutos, Loïc Martínez, and César Montes

Facultad de Informática, Universidad Politécnica de Madrid,
28660 Boadilla del Monte (Madrid), Spain
{falonso, sfrutos, loic, cmontes}@fi.upm.es

Abstract. *Agent-Oriented Software Engineering* has emerged as a powerful engineering discipline that can deal with the complexity of today's software systems (primarily in distributed and open environments) better than other more traditional approaches. However, AOSE does not provide a software development process that naturally leads, if the problem so requires, to an agent architecture. Current agent development methodologies have two separate drawbacks. One is that development processes tend to target an agent organization, which is not necessarily always the best structure, as of the requirements definition stage. The other is that the identification and design of agents are complex, and designer experience plays an essential role in their definition. In this paper, we present the SONIA methodology (Set of mOdels for a Natural Identification of Agents) in an attempt to solve these problems. Based on a generic problem-independent analysis and a bottom-up agent identification process, SONIA naturally outputs an agent-based system.

1 Introduction

Agent-Oriented Software Engineering (AOSE), based on the agent paradigm, has materialized as a powerful technology for developing complex software systems, and it is well suited for tackling the complexity of today's software systems [1]. Having emerged, like so many other disciplines, from Artificial Intelligence, it is now a melting pot of many different computing sciences areas (Artificial Intelligence, Software Engineering, Robotics, and Distributed Computing).

The AOSE concept includes the development of *autonomous software agents* (autonomous elements, with reactive and proactive social ability, trying to accomplish their own task [2]), *multi-agent systems* (MAS) (a set of autonomous agents that interact with each other, each representing an independent focus of system control [3]), and *agent societies* (where the social role of the agents and social laws delimit agent operation [4]).

Agents, MAS and agent societies are now well enough known for researchers and companies to be attracted by the prospects of large-scale agent engineering. The interest they are showing is actually the logical consequence of the successes achieved in this direction, resembling the sequence of events that already took place in other development engineering disciplines (like objects, for example) [5].

M.-P. Gleizes, A. Omicini, and F. Zambonelli (Eds.): ESAW 2004, LNAI 3451, pp. 245 – 260, 2005.

In this paper, we describe the SONIA methodology, an approach for naturally producing a MAS from the system requirements. In section 2, we explain what problems AOSE faces. Section 3 contains an analysis of current agent development methodologies. Section 4 describes the structure of the proposed SONIA methodology and its application to the ALBOR project. Finally, section 5 states the conclusions on the natural development of agents.

2 Problems of AOSE

AOSE is obviously not a panacea, as its use is not always justified. There are problems that an agent approach cannot solve, and others where the outlay and development time required by such an approach would be too costly to be acceptable for companies. We have identified a set of topics to be taken into account when applying AOSE to real problems [6]:

- *Reach agreement on agent theory*. This new paradigm will not be able to expand unless the agent model is standardized with respect to what characteristics define an agent, what types of architecture are available for agents, what agent organizations are possible, what types of interactions there are between agents, etc. Just as UML (Unified Modeling Language) [7] was established to model objects, a modeling language for agents needs to be agreed upon (perhaps AUML [8]).
- *Provide mechanisms for deciding whether the problem should be dealt with using a MAS*. Even if it is initially justified to conceive a multi-agent solution for a given problem, a MAS could turn out to be no good in the end, because, for example, no agents can be identified or there are no interactions between the identified agents.
- *Train development team members in the field of agents and MAS*. A team of developers is not usually familiar with agents and MAS these days, which means that they will have to be trained beforehand in this field if they are to be receptive to such projects and to prevent delays in project development.
- *Provide special-purpose programming languages and development tools*. Although the last few years have seen new languages for programming agent behavior take root, general-purpose languages like Java and C++, etc., are widely used. On the other hand, there are fewer development tools for representing agent structure, and they focus mainly on a particular agent architecture.
- *Use methodologies suited to the development processes*. For organizations to adopt MAS development, the right methodology needs to be provided to guide the team of developers towards the achievement of objectives, without this requiring in-depth training in this field. A critical stage in the development of a MAS is the selection of the methodology to be followed. A good methodology should provide the models for defining the elements of the multi-agent environment (agents, objects and interactions) and the design guidelines for identifying these elements, their components and the relationships between them.

As regards the question of methodology, a wide variety of *methodological proposals* have emerged for AOSE development [9][10][11][12]. Although they have

all played an important role in establishing this field, they do not provide suitable mechanisms for formulating a *natural* process for developing a MAS system or an agent society from system requirements. That is, the gradual discovery and identification of concepts, relationships, tasks, knowledge, behaviors, objects, agents, MAS and agent societies from the problem statement. Additionally, a good methodology should not force a given architecture (object-oriented, agent-oriented, etc.) upon developers from the beginning. It is the system specifications analysis that should point developers towards the best suited architecture for solving the problem.

Based on research and development efforts in the field of AOSE, we think that an agent-oriented development methodology should have the following features [6]:

- *It should not condition the use of the agent paradigm right from analysis*. It is too risky to decide whether the system is to be designed using a multi-agent architecture in the analysis or conceptualization phase, as the problem is not fully specified at this early stage of development. It is not until the design phase that enough is known about the problem specifications and architecture to make this decision.
- *It should naturally lead to the conclusion of whether or not it is feasible to develop the system as a MAS*. At present, it is the developer who has to decide, based on his or her expertise, whether or not to use a MAS to solve the problem. Because of its high cost, this is a tricky decision that cannot be made using heuristics. Note that, depending on the application domain, design and implementation using a multi-agent architecture may have a high development cost (time, money and resources), apart from calling for experienced personnel. On the other hand, the modularity of multi-agent systems may improve development costs.
- *It should systematically identify the components of a MAS*. Current methodologies leave too much to the designer with respect, for example, to agent identification. Designer experience is therefore vital for producing a quality MAS.

  Component-driven bottom-up agent identification is the most objective criterion, as it depends exclusively on the problem and eases the systematization and automation of the identification process. On the other hand, the role (or actor)-driven criterion is more subjective, as roles or actors depend on the analyst/designer who identifies them.
- *If the problem specifications call for an agent society, it should naturally lead to this organizational model*. The development of a software system using a reductionist, constructivist or agent society architecture should be derived from the problem specifications, which will lead to the best suited architecture. Current agent-oriented methodologies focus on the development of the actual agent architecture (internal agent level) and/or its interactions with other MAS agents (external agent level), but very few cover the concept of social organization.
- *It should produce reusable agents, should be easy to apply and not require excessive knowledge of agent technology*. The concept of reuse has been one of the biggest contributions to software development. The provision of libraries has furthered procedure-, object-, or component-oriented engineering. For this advance to take place in AOSE, agent components (interaction protocols, etc.) need to be reusable and easy to use. Current agent-oriented design methodologies and methods do not account for reusable systems and call for high proficiency in MAS

technology for use. As MAS technology is related to many disciplines (artificial intelligence, psychology, sociology, economics, etc.), intensive knowledge of agent technology is required. This relegates the design of these systems to universities, research centers and companies with the latest technology.

The specific characteristics of MAS and MAS development-related problems indicate that agent-based problem solving cannot be dealt with intuitively. It calls for a methodological process that naturally leads to the use of agents in problem solving.

3 Analysis of Current Agent Development Methodologies

On account of the advance in agent technology over the last ten years, several methodologies have emerged to drive MAS development [9][10][11][12]. These methodologies are classed according to the discipline on which they are based (Fig. 1):

- *Agent Technology-Based Approaches*: they focus on social level abstractions, like the agent, group or organization.
- *Object Orientation-Based Approaches*: they are characterized by extending object-oriented techniques to include the notion of agency.
- *Knowledge Engineering-Based Approaches*: they are characterized by emphasizing the identification, acquisition and modeling of knowledge used by the agent components.

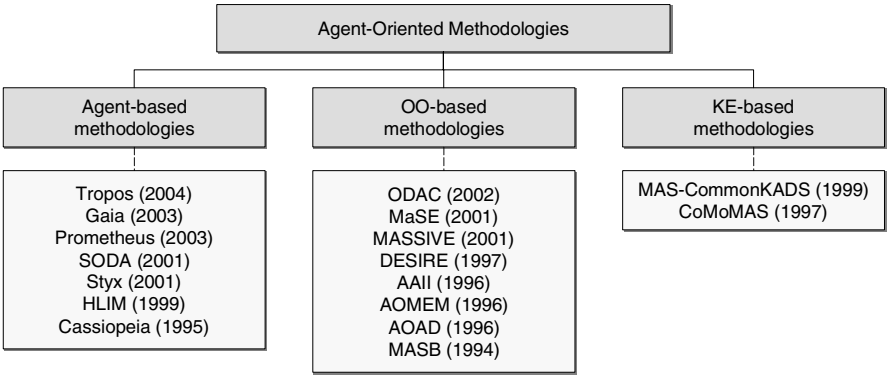

Fig. 1. Agent-Oriented Methodologies

3.1 Agent Technology-Based Methodologies

Agent Technology-Based Methodologies focus on social level abstractions, like the agent, group or organization.

The most representative methodologies are: Tropos [13], Gaia [14], Prometheus [15], SODA [16], Styx [17], HLIM [18] and Cassiopeia [19]. Table 1 describes the most significant methodological aspects of agent technology-based methodologies for our analysis.

Table 1. Agent Technology-Based Methodologies

agent paradigm selection	specification or analysis phase (all)	design phase (none)
agent identification process	role-driven top-down (all)	component-driven bottom-up (none)
MAS aspects	intra- & inter-agent (Tropos, Gaia, Prometheus, Styx, HLIM)	social structure (SODA, Cassiopeia)
environment analysis	environment (SODA)	objects (Tropos, Prometheus, Styx, SODA)

Although this methodological line is gaining in importance in agent development, the methodologies suffer from some limitations on key points:

- These methodologies propose the use of the agent paradigm as of the specification (Prometheus, HLIM, Cassiopeia) or analysis (Tropos, Gaia, SODA, Styx) phases. The choice of a multi-agent system should be a *design* decision. Therefore, a good agent-oriented methodology should not conduct a specific agent-oriented analysis. None of the methodologies account for the use of a generic analysis model that can be used to evaluate whether or not a multi-agent approach is suitable.
- All of the methodologies identify agents from social roles (Gaia, SODA, Styx, HLIM, Cassiopeia) or actors (Tropos, Prometheus) following a top-down identification process and none from their components.
- Three aspects need to be dealt with to develop a MAS: intra-agent structure, inter-agent structure and social structure. Most of the methodologies cover the intra-agent and inter-agent aspects (Tropos, Gaia, Prometheus, Styx, HLIM), but only SODA and Cassiopeia account for social structure.
- The analysis of the environment is a key point. SODA is the only methodology to analyze the environment, its entities and their interactions.

3.2 Object Orientation-Based Methodologies

Object Orientation-Based Methodologies are characterized by extending object-oriented techniques [20] to include the notion of agency.

The most representative methodologies are: ODAC [21], MaSE [22], MASSIVE [23], DESIRE [24], AAII [25], AOMEM[26], AOAD[27] and MASB[28]. Table 2 lists which of the examined methodological features object orientation-based methodologies have.

Table 2. Object Orientation-Based Methodologies

agent paradigm selection	specification or analysis phase (ODAC, AOAD)	design phase (MaSE, MASSIVE, DESIRE, AAII, AOMEM, MASB)
agent identification process	role-driven top-down (ODAC, MaSE. MASSIVE, AAII, AOMEM, AOAD, MASB)	component-driven bottom-up (DESIRE)
MAS aspects	intra- & inter-agent (ODAC, MASB, DESIRE, AAII, AOMEM, AOAD, MASB)	social structure (MASSIVE, AOAD)
environment analysis	environment (MASSIVE)	objects (ODAC, MASB)

From the viewpoint of correct agent orientation, this methodological line is beset by the following problems. It does not account for the use of a generic analysis model. Some methodologies (ODAC and AOAD) identify agents during analysis. Only the DESIRE methodology implements a proper component-driven bottom-up agent identification process. Almost all the methodologies (ODAC, MASB, DESIRE, AAII, AOMEM, AOAD and MASB) cover the intra-agent and inter-agent aspects, but only MASSIVE and AOAD cover the social structure. Finally, with the exception of MASSIVE, none of the methodologies takes into account the environment features.

These methodologies treat agents like complex objects, which is wrong, because agents have a higher level of abstraction than objects. They also fail to properly capture the autonomous behavior of agents, interactions between agents, and organizational structures [17].

3.3 Knowledge Engineering-Based Methodologies

Knowledge Engineering-Based Methodologies are characterized by emphasizing the identification, acquisition and modeling of knowledge used by the agent components.

The most representative methodologies originate from the CommonKADS methodology [29] are MASCommonKADS [30] and CoMoMAS [31]. Table 3 lists the features of these methodologies for our analysis.

Table 3. Knowledge Engineering-Based Methodologies

agent paradigm selection	specification or analysis phase (MAS-CommonKADS)	design phase (CoMoMAS)
agent identification process	role-driven top-down (MAS-CommonKADS)	component-driven bottom-up (CoMoMAS)
MAS aspects	intra- & inter-agent (all)	social structure (none)
environment analysis	environment (none)	objects (none)

These methodologies also present some problems. Like the other approaches described earlier, these methodologies do not account for the use of a generic analysis model. MAS-CommonKADS identifies agents during analysis, following a role-driven top-down process (identifying actors). Both of them account for the intra-agent and inter-agent aspects, but do not cover social issues or analysis of the environment.

3.4 Analysis of Current Agent Development Methodologies

The methodological approach based directly on agent technology is perhaps better than the other two, because it is based on the intrinsic concept of agent and agent organization in a MAS. It basically falls down on the point that it confines problem analysis to the agent paradigm, whereas this paradigm may turn out to be unsuitable if agent technology is not a good option for dealing with the problem in question.

Briefly, we believe that a good AOSE methodology is one that defines an architecture-independent *generic analysis model* and a *design model* that can

systematically identify agents following a component-driven bottom-up agent identification process, can identify the intra-agent, inter-agent and social structure of the system, can analyze the environment and can identify environment objects.

4 SONIA Methodology

The SONIA (Set of mOdels for a Natural Identification of Agents) methodology [6] allows the generation of a multi-agent architecture to solve a problem (whose conceptualization is not conditioned by the agent paradigm) according to a *Multi-Agent Design Model* that systemizes and automates the activities of identifying the MAS components.

The phases and stages of which the SONIA methodology is composed are listed below, along with the models generated in each stage (Fig. 2):

- *Conceptualization:* The problem is analyzed on the basis of the problem statement using an analysis model that does not condition the design paradigm. The result is an initial *Structural Model*, which describes the overall structure of the domain and an initial *Task Model*, which describes how to solve problems occurring in the domain.
- *Extended Analysis*: The above models are refined and expanded to include the features of the environment and the external system entities, producing the following models: an *Environment Model*, which defines the external system entities and system interactions with these entities; a *Structural Model*, which includes domain knowledge structures of the external system entities that interact with the system; and a *Task Model*, which adds the functionalities required for interaction with the external system entities.

  The Conceptualization and Extended Analysis stages form the MAS analysis phase.
- *Synthesis*: This stage is aimed at improving the identification of agents from their components. For this purpose, the elements of the Structural and Task Models are grouped depending on concepts that are characteristic of agents such as knowledge, behaviors and responsibilities.

  This stage provides a smooth transition from analysis to design, outputting: a *Knowledge Model*, which identifies the knowledge components inherent to the problem by grouping concepts and associations from Structural Model; a *Behavior Model*, produced by grouping tasks, subtasks and methods from the Task Model; and a *Responsibility Model*, output by establishing the relationships between knowledge components and behaviors.
- *Architectural Design*: In this stage, we decide whether or not the system will be designed following a multiagent architecture. If a MAS is designed, the entities of the architecture are also defined.

  The generated models are: an *Agent Model*, which identifies and defines what elements should be designed as autonomous agents; an *Object Model*, which identifies and defines what passive elements there are in the environment; and an

Interaction Model, which identifies and defines the relationships among agents and between agents and objects.

The stages of Synthesis and Architectural Design are what make up the design phase.

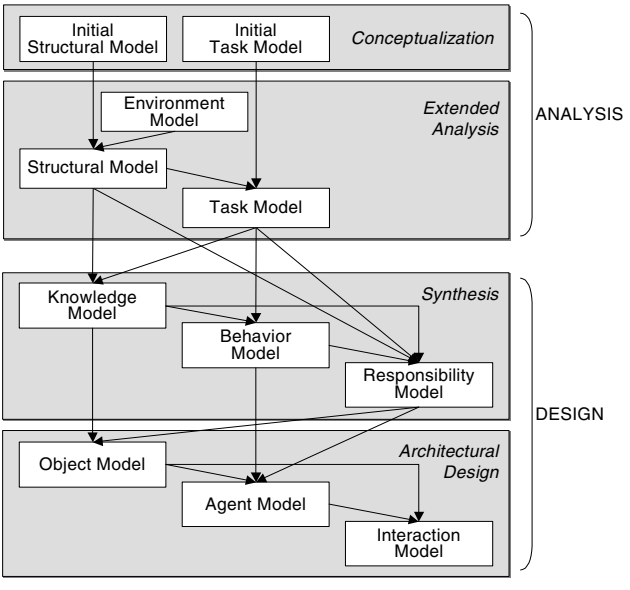

Fig. 2. Phases of the SONIA methodology

Although the methodological process is top-down, this methodology follows a bottom-up process to build the MAS architecture. Instead of identifying the MAS entities and then the components of these entities, the methodology starts by identifying the atomic elements (concepts, associations, tasks, etc.) output by system analysis, which are then grouped into more complex elements (components), from which the agents and objects of the MAS architecture will be able to be identified. This makes the generated system highly extensible and facilitates agent and component extension, modification and reuse.

In the following, the phases and stages of the SONIA methodology and their application to the development of the ALBOR project (Barrier-Free Computer Access) are briefly described [32][33].

ALBOR was conceived as an Internet-based intelligent system designed to provide guidance on the evaluation of disabled people's computer access skills and on the choice of the best suited assistive technologies.

Each system session is divided into four stages:

1. *User identification*: user personal particulars and other information are collected in order to start the session.

2. *Session preparation*: the user is informed about the goals of the questionnaire, how the session will be performed and whether any preliminary training is necessary.
3. *Survey taking*: the user is asked a series of questions, which will be depend on responses to questions already answered and will be confined to the questions strictly necessary for the evaluation of the person in question.
4. *Result evaluation*: an evaluation report with several recommendations for the user to decide which is best suited for her/him is sent to the user.

4.1 Analysis

The elicited requirements are analyzed using the Set Theory Based Conceptual Model (SETCM) [33][34], an analysis method that was defined to achieve several goals. First, the method is design independent: it uses terminology other than design languages to give a real understanding of the problem under analysis. Second, SETCM is able to analyze problems of different kinds, ranging from the simpler, algorithmic problems to more complex and knowledge-based problems. Third, the method has a solid formal foundation, thanks to which it can unambiguously represent the results of the analysis. Fourth, SETCM includes a comprehensive and easy-to-understand textual notation, which is a deterrent to the use of mathematical notations. Finally, the method includes a graphical notation, which eases the understanding of large models.

SETCM is design independent and capable of analyzing complex problems thanks to the fact that the SETCM modeling elements were carefully chosen and defined. These elements were selected from the elements commonly used in other approaches, eluding design-specific terms and incorporating new elements where necessary. Some of these elements are concepts, associations, attributes, classifications, tasks and task-methods. The elements were defined using Set Theory vocabulary, which is the basis of mathematics. For instance, an association is a subset of the Cartesian product of the elements involved. The SETCM elements are grouped into two components: the *Structural Model*, which represents the structure of a domain (elements and relationships between them) and the states that can occur within this domain, and the *Task Model*, representing domain problem solving.

To achieve the goal of establishing a formal foundation, all the modeling primitives were formalized using the main elements of Cantor's naïve set theory, while defining a rigid modeling structure that eludes the contradictions of this theory. Thus, SETCM has a formal modeling core (with more than 700 formalized symbols). This core contains a large set of formal primitives that can be added to in the future by defining and formalizing new elements based on existing components.

The last two goals (textual and graphical notations) are concerned with resolving pragmatic issues. The textual notation represents all the SETCM modeling primitives, is a substitute for the use of mathematics and is highly readable. The graphical notation is based on UML using stereotypes and eases the understanding of large quantities of information, reduces the apparent complexity of the analytical models and is more expressive than the textual notation [34].

SETCM has been applied to develop real systems, which were finally designed using a variety of paradigms (structured, object-oriented, knowledge-based) and even a combination of paradigms.

As mentioned earlier, the Initial Structural Model and the Initial Task Model of SONIA are built using SETCM. These models are refined and expanded to capture the system Environment and External Entities, successively producing:

- An *Environment Model*, which defines the system external entities and their interactions with the system.
- A *Structural Model*, which includes structures from the knowledge domain of the external entities that interact with the system.
- A *Task Model*, which adds the functionalities required to interact with the system external entities defined in the Environment Model.

4.2 Design of the Multi-agent Architecture

The Analysis phase is followed by the Multi-Agent Architecture Design, which is divided into two stages: Synthesis and Architectural Design.

The *Synthesis* stage allows the component-driven identification of agents (bottom-up process) in the Multi-Agent Architecture Design stage. The elements of the Structural Model and Task Model are grouped depending on characteristics of agents, such as knowledge, behaviors and responsibilities, outputting the following models:

- A *Knowledge Model*, which identifies the knowledge components by grouping Structural Model concepts and associations. These groupings are identified because the internal cohesion of their members is high, coupling with other groupings is low and they are used to perform tasks of the same behaviors. The knowledge components will be used internally or shared by the agents.

  The groupings resulting from the first version of the model only check for high cohesion and low coupling among their members. The final version will be built when the responsibilities between knowledge components and behaviors (Responsibility Model) are established and will also check that the members of the groupings are used to do the same tasks.
- A *Behavior Model*, produced by grouping Task Model tasks, subtasks and methods. The behaviors will be part of the agents. These groupings are identified because their tasks and subtasks depend on each other through their methods and they use the same knowledge components in problem solving.

  The groupings from the first version of the model only check for the dependence of some tasks on others through task methods. The final version, which is built when the responsibilities between knowledge components and behaviors (Responsibility Model) are established, will also check that they use the same knowledge in problem solving.
- A *Responsibility Model*, output by relating knowledge components to behaviors. The purpose of this model is to be able to identify agents and environment objects.

  A key activity during the design of this model is to refine the Knowledge and Behavior Models to meet all the conditions.

The *Architectural Design* stage focuses on the definition of the architectural components by means of the following models: Agent Model, Object Model and Interaction Model.

Not until the Agent Model is built is a decision made as to whether the architecture can be implemented by means of agents or a different paradigm needs to be used. This choice is chiefly based on whether or not agents can be identified. For an entity to be able to considered as an autonomous agent, it should have a behavior and the right knowledge components to perform the tasks of this behavior, have at least one defined goal and one utility, and perceive and act in the environment.

If no agents can be identified, another design paradigm will have to be chosen. One possible alternative would be an object-oriented design, reusing objects and interactions identified in the multi-agent architecture design stage. Another possibility would be to design the system as a knowledge-based system, reusing the knowledge components, behaviors and responsibilities output in the synthesis stage.
The Architectural Design models are:

- An *Agent Model*, which identifies and defines, from the Responsibility, Knowledge and Behavior Models, what entities should be designed as autonomous agents. An agent is identified because it is an environment-sensitive entity (it perceives and acts in the environment) that has knowledge to bring into play its behaviors in pursuit of goals and is activated when its utilities are required.

  Therefore, *knowledge* is groupings of concepts and associations that the agent uses to reason and *behaviors* are groupings of tasks that allow the agent to develop the function for which it was conceived. The result of executing a behavior can affect the environment objects or its internal knowledge.

  Goals are objectives pursued by the agent. The agent will execute behaviors to achieve its goals. *Utilities* are triggers that activate the agent. The agent will assess the execution of some of its behaviors if their utilities are met. Goals and utilities are logical conditions on the state of the environment objects or on the state of their internal knowledge.

  Sensors listen to the environment objects and notify the agent every time a change takes place in the objects they are listening in on. This notification can cause some of the agent's utilities or goals to be met. *Actuators* modify environment objects, and the agent will use the respective actuator every time it needs to modify an environment object during behavior execution.
- An *Object Model*, which identifies and defines, from the Responsibility, Knowledge and Behavior Models, what passive elements are part of the environment. These objects are knowledge components identified during the synthesis phase. The main feature of an object is that the knowledge of this object is responsible for more than one behavior or, in other words, is shared by several behaviors. Access to objects will be divided by *levels*, and the knowledge components that are accessed by the same behavior tasks will be grouped at the same level.
- An *Interaction Model*, which identifies and defines what relationships there are in the system among agents and between agents and objects.

  Agent-agent relations occur when both agents interact to take any particular action. This interaction takes place according to interaction protocols based on *speech act theory* [35]. In the case of a reductionist MAS system (designed by one

and the same person), the interaction protocol is designed at the same time as the actual agent. In the case of a constructivist MAS system (designed by different people), the interaction protocols are located in a library and are accessed by the agents at interaction time. *Agent-object relations* occur when an agent accesses an object level, either through a sensor or an actuator.

This architecture accounts for the two communication types: *asynchronous communication*, using environment objects to subscribe to events of interest to the agent; and *synchronous communication*, through protocols contained in the Interaction Model.

4.3 Design of the ALBOR System

Fig. 3 shows how the Analysis, Synthesis and Architecture Models are built. For simplicity's sake, it shows only the concepts and associations that are the source of the "Questionnaires" knowledge component, and tasks and methods that are the source of the "TakeSurvey" behavior.

The concepts and associations gathered in the Analysis phase were synthesized as knowledge components using a technique based on Kelly's constructs [36], and the tasks and methods as behaviors using heuristics applied to task decomposition and task dependencies. These techniques, used to output the knowledge components and behaviors, assure highly coherent and low-coupled groupings. Then the responsibilities between knowledge components and behaviors were established from the relationships of concept/association used in task/subtask. These responsibilities lead to changes in the Knowledge and Behavior Models. The models are modified according to knowledge and behavior grouping/division rules based on the cardinalities of the relationships of concept/association used in task/subtask. The Knowledge, Behavior and Responsibility Models are the final result of the synthesis.

It is not until the Agent Model is built that a decision is made as to whether the architecture can be implemented by means of agents or a different paradigm needs to be used. This choice is chiefly based on whether or not agents can be identified. For an entity to be able to be considered as an autonomous agent, it should have a behavior and the right knowledge components to perform the tasks of this behavior, have at least one defined goal and one utility, and perceive and act in the environment.

To complete the multi-agent architecture design phase, the environment agents and objects were identified. The objects were identified from the Responsibility Model, and the knowledge shared by several behaviors was chosen as environment objects. Following this criterion, we identified the "Users", "External" and "Media" objects. Agents were also identified from responsibilities. Again, agents should have a behavior, knowledge components, goals and utilities, and sensors and actors. For example, the responsibility between "Questionnaires" knowledge and "TakeSurvey" behavior produces "Survey-Taker". The Agent, Object and Interaction Models are the final result of the architecture design stage.

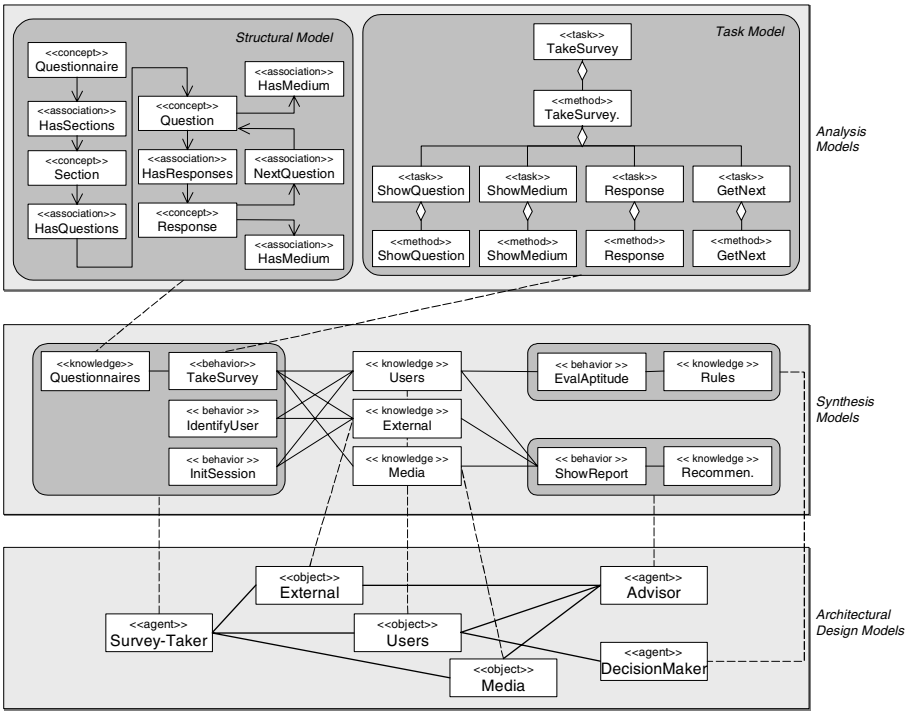

Fig. 3. ALBOR: From analysis models to architectural design models

5 Conclusions

AOSE is unquestionably a very good technique for solving complex problems, especially in distributed, open and heterogeneous environments. For this technology to be routinely used in companies like object-oriented approaches are, there is a need for mechanisms suited for deciding whether or not the problem should be solved using agents. Also the identification and design of agents should be a natural and straightforward process that does not require a lot of expertise so that there is no obstacle to its application by developers. Although they have made a big contribution to improving AOSE, current agent development methodologies do not satisfactorily solve the above-mentioned problems.

In this paper, we have pointed out some features that an agent-oriented development methodology should have and detailed which of these features are missing from the most important methodologies used within the agent paradigm. Also, we have presented an overview of the SONIA methodology, illustrated by the ALBOR case study, which includes these features and naturally leads from requirements elicitation to MAS and agent-based development.

References

1. Zambonelli, F., Jennings, N. R., Omicini, A., Wooldridge, M.: Agent-Oriented Software Engineering for Internet Applications. In: Omicini, A., Zambonelli, F., Klusch, M., Tolksdorf, R. (eds.): Coordination of Internet Agents: "Models, Technologies and Applications". Springer-Verlag (2001) 326-346
2. Huhns, M., Singh, M. P. (eds.): Readings in Agents. Morgan Kaufmann, San Mateo, CA. (1998)
3. Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley & Sons, LTD (2002)
4. Epstein, J. M., Axtell, R. L.: Growing Artificial Societies: Social Science from the Bottom Up. The Brooking Institution Press & The MIT Press (1996)
5. Lind, J.: Issues in Agent-Oriented Software Engineering. In: Ciancarini, P., Wooldridge, M. (eds.): Agent-Oriented Software Engineering, LNAI 1957. Springer-Verlag (2001) 45-58
6. Frutos, S.: Modelo de Diseño de una Arquitectura Multi-Agente Basado en un Modelo de Sociedad de Agentes (*Multi-Agent Architecture Design Model based on an Agent Society Model*). PhD Thesis. Universidad Politécnica de Madrid, Spain (2003)
7. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley Longman (1999)
8. Odell, J., Parunak, H. V. D., Bauer, B.: Extending UML for Agents. In: Wagner, G., Lesperance, Y., Yu, E. (eds.): Proc. of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence. ICue Publishing (2000)
9. Weiss, G.: Agent Orientation in Software Engineering. Knowledge Engineering Review, Vol. 16(4) (2002) 349-373
10. Wooldridge, M., Ciancarini, P.: Agent-Oriented Software Engineering: The State of the Art. In: Ciancarini, P., Wooldridge, M. (eds.): Agent-Oriented Software Engineering, LNAI 1957. Springer-Verlag, Berlin (2001) 1-28
11. Tveit, A.: A Survey of Agent-Oriented Software Engineering. First NTNU CSGSC (2001)
12. Iglesias, C.A., Garijo, M., González, J.C.: A Survey of Agent-Oriented Methodologies. In: Müller, J.P., Singh, M. P., Rao, A. (eds.): Intelligent Agents V (ATAL'98), LNAI 1555. Springer-Verlag, Berlin (1999) 317-330
13. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An Agent Oriented Software Development Methodology. Int. Journal of Autonomous Agent and MultiAgent System, Vol. 8(3) (2004) 203-236
14. Zambonelli, F., Jennings, N. R., Wooldridge, M.: Developing Multiagent Systems: The Gaia Methodology. ACM Transactions on Software Engineering and Methodology, Vol. 12(3) (2003) 317-370
15. Padgham, L., Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents. In: Giunchiglia, F., Odell, J., Weiss, G. (eds.): Agent-Oriented Software Engineering III, LNCS 2585. Springer-Verlag. Berlin (2003) 174-185
16. Omicini, A.: SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems. In: Ciancarini, P., Wooldridge, M. (eds.): Agent-Oriented Software Engineering, LNAI 1957. Springer-Verlag. Berlin (2001) 185-194
17. Bush, G., Cranefield, S., Purvis, M.; The Styx Agent Methodology. The Information Science Discussion Paper Series, Number 2001/02. University of Otago. New Zealand (2001)

18. Elammari, M., Lalonde, W.: An Agent-Oriented Methodology: High-Level and Intermediate Models. Proc. of the First Bi-Conference. Workshop on Agent-Oriented Information Systems (AOIS'99). Heidelberg, Germany (1999)
19. Collinot, A., Carle, P., Zeghal, K.: Cassiopeia: A Method for Designing Computational Organizations. Proc. of the First Int. Workshop on Decentralized Intelligent Multi-Agent Systems. Krakow, Poland (1995) 124-131
20. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison Wesley Longman. Reading, MA (1999)
21. Gervais, M.: ODAC: An Agent-Oriented Methodology Based on ODP. Journal of Autonomous Agents and Multi-Agent Systems, Vol. **7**(3) (2002) 199-228
22. Wood, M. F., DeLoach, S. A.: An Overview of the Multiagent Systems Engineering Methodology. In: Ciancarini, P., Wooldridge, M. (eds.): Agent-Oriented Software Engineering, LNAI 1957. Springer-Verlag, Berlin (2001) 207-222
23. Lind, J.: Iterative Software Engineering for Multiagent Systems: The MASSIVE method, LNCS- 1994. Springer-Verlag (2001)
24. Brazier, F. M. T., Dunin-Keplicz, B., Jennings, N., Treur, J.: Desire: Modeling Multi-Agent Systems in a Compositional Formal Framework. Int. Journal of Cooperative Information Systems, Vol. 6. Special Issue on Formal Methods in Cooperative Information Systems: Multiagent Systems (1997)
25. Kinny, D., Georgeff, M., Rao, A.: A Methodology and Modeling Technique for Systems of BDI Agents. In: van de Velde, W., Perram, J. W. (eds.): Agents Breaking Away (MAAMAW'96), LNAI 1038. Springer-Verlag, Berlin (1996) 56-71
26. Kendall, E. A., Malkoun, M. T., Jiang, C. H.: A Methodology for Developing Agent Based Systems. In: Zhang, C., Lukose, D. (eds.): Distributed Artificial Intelligence - Architecture and Modeling, LNAI 1087. Springer-Verlag, Germany (1996) 85-99
27. Burmeister, B.: Models and Methodology for Agent-Oriented Analysis and Design. In: Fischer, K. (ed.): Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems, Saarbrücken, Germany (1996)
28. Moulin, B., Cloutier, L.: Collaborative Work Based on Multi-Agent Architectures: A Methodological Perspective. In: Aminzadeh, F., Jamshidi, M. (eds.): Soft Computing: Fuzzy Logic, Neural Networks and Distributed Artificial Intelligence. Prentice-Hall, N.J., USA (1994) 261-296
29. Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., Van de Velde, W., Wielinga, B.: Knowledge Engineering and Management. The CommonKADS Methodology. The MIT Press. Cambridge, MA (1999)
30. Iglesias, C.A., Garijo, M., González, J.C., Velasco, J. R.: Analysis and Design of Multiagent Systems using MAS-CommonKADS. In: Singh, M. P., Rao A. S., Wooldridge, M. (eds.): Intelligent Agents IV: Agent Theories, Architectures, and Languages (ATAL97), LNAI 1365. Springer-Verlag, Berlin (1999) 313-326
31. Glaser, N.: The CoMoMAS Methodology and Environment for Multi-Agent System Development. In: Zhang, C., Lukose, D. (eds.): Multi-Agent Systems - Methodologies and Applications, LNAI 1286. Springer-Verlag, Berlin (1997) 1-16
32. Alonso, F., Barreiro, J. M., Frutos, S., Montes, C.: Multi-Agent Framework for Intelligent Questionnaire on the Web. Proc. of the Third World Multiconference on Systemics, Cybernetics and Informatics (SCI-99) and the Fifth Int. Conference on Information Systems Analysis and Synthesis (ISAS'99), Vol. III. Orlando, USA (1999) 8-15

33. Alonso, F., Frutos, S., Fuertes, J. L., Martínez, L. A., Montes, C.: ALBOR. An Internet-Based Advisory KBS with a Multi-Agent Architecture. Int. Conference on Advances in Infrastructure for Electronic Business, Science, And Education on the Internet (SSGRR 2001), L'Aquila, Italy (2001) 1-6
34. Martínez, L.A.: Método para el Analysis Independiente de Problemas (*Method for Independent Problem Analysis*). PhD Thesis. Universidad Politécnica de Madrid. Spain (2003)
35. Austin, J.L.: How to Do Things with Words. Harvard University Press. Cambridge, MA (1962)
36. Kelly, G. A.: The Psychology of Personal Constructs. Norton (1995)