1. INTRODUCTION

Agent-based computing has materialized as a powerful technology for developing complex software systems [1]. Having emerged, like so many other disciplines, from artificial intelligence, it is now a melting pot of many different research areas (artificial intelligence, software engineering, robotics, and distributed computing).

Agent-Oriented Software Engineering (AOSE) stems from a line of research including the autonomous software agent as an autonomous element, with reactive and proactive social ability, trying to accomplish its own task [2], the multiagent system (MAS) as 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]. AOSE is well suited for tackling the complexity of today’s software systems [5].

This line of research resulted in a new paradigm: the agent paradigm. Matured in research laboratories by developing resources and applications, with the appearance of architectures, models and methodologies, this paradigm is now acquiring the status of an engineering discipline.

This effort to convert agent-oriented system development into a genuine engineering discipline capable of large-scale MAS development has led to a variety of methodological proposals [6–8]. Although they all have played an important role in establishing this field, they have omissions that are an obstacle to the formulation of a natural process for developing an MAS system or an agent society from system requirements. 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.

In this paper, we describe a methodological approach for naturally producing an MAS or agent society from the system requirements. In section 2, we explain what problems MAS developers face. Section 3 details the omissions of current MAS development methodologies. Section 4 describes the structure of the proposed SONIA methodology. Section 5 outlines a case study: the ALBOR project. Finally, section 6 states the conclusions on the natural development of MAS.
2. PROBLEMS OF DEVELOPING A MULTIAGENT SYSTEM

From the pragmatic viewpoint, an MAS is a computer system composed of a set of agents capable of interacting with each other and with the environment in which they are located to achieve their own goals.

An MAS can be described by abstracting three underlying concepts:

- **Agents**: active system entities with the ability to sense and act in the system environment.
- **Environment** (organizational relationships): where the mutual dependencies and social relationships between agents are defined. It provides the conceptual framework for each agent to find a well-defined position (or role) in a society that stimulates and structures the interactions between agents.
- **Interaction**: this is the link element between the agents and between the agents and the objects in the environment.

The MAS-based approach is obviously not a panacea, as its use is not always justified. There are problems where the outlay and development time required by such an approach would be too costly to be acceptable for companies. It is worthwhile to employ a multiagent architecture basically: when the environment is open, dynamic, uncertain or complex, when agents are the natural metaphor, when data, control or expertise is distributed, or when legacy systems need to be integrated [9,10].

Nevertheless, there are a series of problems and challenges that need to be dealt with to develop quality agent-based software. Wooldridge and Jennings [11] identified seven categories of potential problems within AOSE. Apart from these, we have identified a set of methodological topics to be taken into account when applying AOSE to real problems [12]:

- **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) [13] was established to model objects, a modeling language for agents needs to be agreed upon (perhaps AUML [14]).
- **Provide mechanisms for deciding whether the problem should be dealt with using an MAS**. Even if it is initially justified to conceive a multiagent solution for a given problem, an 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**. An organization’s 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**. Over the last few years, new languages for programming agent behavior have taken root, although general-purpose languages, like Java and C++, have also been widely used. On the other hand, there are fewer development tools for representing agent structure, and they focus mainly on a particular type of agent architecture and on some specific interaction protocols.
- **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 requiring in-depth training in this field. A critical stage in the development of an MAS is the selection of the methodology to be followed. A good methodology should provide the models for defining the elements of the multiagent environment (agents, objects and interactions) and the design guidelines for identifying these entities, their components and the relationships between them.

Based on research and development efforts conducted in the field of AOSE, we believe that an agent-oriented development methodology should have the following features [12]:

- **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 multiagent 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 an MAS**. At present, it is the developer who has to decide, based on his or her expertise, whether or not to use an MAS to solve the problem. 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 reduce development costs.
- **It should systematically identify the elements of an MAS**. Current methodologies leave too much to the designer with respect, for example, to agent identification. Although designer experience is vital for producing a quality MAS, we believe that the actual methodology should assist with identification.
- **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 [15], constructivist [16] or agent society [4] 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 facilitate the reusability of agents, if possible**. 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. We regard this as being a feature...
that is hard to achieve because of the complexity of agent architectures, but, for AOSE to advance and establish itself, its elements (agent components, interaction protocols, etc.) need to be reusable and easy to use. Very few agent-oriented methodologies produce reusable elements, such as capabilities in Prometheus and components in DESIRE.

- It should be easy to apply and not require excessive knowledge of agent technology. Current agent-oriented design methodologies and methods call for a high level of MAS technology proficiency for use. As MAS technology is related to many disciplines (artificial intelligence, psychology, sociology, economics, etc.), the design of these systems would be relegated to universities, research centers and companies with the latest technology.

The specific characteristics of multiagent systems and multi-agent system 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. Omissions 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 [6–8]. These methodologies are classed according to the discipline on which they are based:

- **Agent Technology-Based Approaches**: these methodologies focus on social level abstractions, like the agent, group or organization.

- **Object Orientation-Based Approaches**: these methodologies are characterized by extending object-oriented techniques [34] to include the notion of agency.

- **Knowledge Engineering-Based Approaches**: these methodologies are characterized by emphasizing the identification, acquisition and modeling of knowledge used by the agent components. These methodologies originate from the CommonKADS [35] methodology.

Figure 1 shows the most commonly referenced methodologies of these classes. The problems that plague these methodologies are as follows:

- The agent technology-based methodologies propose the use of the agent paradigm as of the specification (Prometheus, HLIM, Cassiopeia) or analysis (Tropos, Gaia, SODA, Styx) phases, which they use as a starting point for design. The object orientation-based methodologies, however, identify agents in the design phase, and the knowledge engineering-based methodologies identify agents as of analysis (MAS-CommonKADS) or design (CoMoMAS). 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. Indeed, an analysis that is independent of the design paradigm and can decide what design (multiagent or otherwise) is the best would be preferable.

- Most methodologies identify agents from social roles (Gaia, SODA, Styx, HLIM, Cassiopeia, ODAC, MaSE, MASSIVE, AAII, AOMEM, AOAD, MASB, MAS-CommonKADS) or from actors (Tropos), but very few do so from their components (or parts of the agent) (Prometheus, DESIRE, CoMoMAS). In the role (or actor)-driven bottom-up agent identification process, agents are identified first from roles or actors (high level) and then from their components (knowledge, behaviors, etc.) and interactions with other agents (low level). This criterion is more subjective, as roles or actors depend on the analyst/designer who identifies them. On the other hand, component-driven top-down agent identification is a more objective criterion, as it depends exclusively on the problem and eases the systematization and automation of the identification process.

- Three aspects need to be dealt with to develop an MAS: internal agent design (intra-agent structure), design of interactions with other agents (inter-agent structure) and design of the structure of organizations or societies in which the agents can participate (social structure). Most of the methodologies cover the intra-agent and inter-agent aspects (Tropos, Gaia, Prometheus, Styx, HLIM, ODAC, MASB, DESIRE, AAII, AOMEM, AOAD, MASB, MASCommonKADS, CoMoMAS), but only Gaia, SODA, Cassiopeia, MASSIVE, and AOAD account for social structure.

- The analysis of the environment and the identification of objects of the environment is a key point for examining problems that are intended to be solved using a multiagent architecture. Only Gaia, SODA, and MASSIVE analyze the environment, its entities and their interactions. Moreover, only a few methodologies explicitly identify environment objects (Tropos, Prometheus, Styx, SODA, ODAC, MASB).

The methodological approach based directly on agent technology is perhaps better than the other two approaches, because it is based on the intrinsic concept of agent and agent organization in an 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 should:

- define an architecture-independent generic analysis model,
- define a design model that can systematically identify
agents following a component-driven bottom-up process,
• identify the intra-agent, inter-agent and social structure of
  the agent, and
• analyze the environment and identify environment objects.

4. SONIA METHODOLOGY

The SONIA (Set of mOdeIs for a Natural Identification of Agents) methodology [36] generates a multiagent architecture to solve a problem according to a Multiagent Design Model that systemizes and automates the activities of identifying the MAS elements. SONIA conceptualization is not conditioned by the agent paradigm. Likewise, the methodology defines an agent society that flexibly and dynamically facilitates problem solving and can be used to integrate indispensable legacy systems.

The phases and stages of which the SONIA methodology is composed are listed below, along with the models generated in each stage (Figure 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 and an initial Task Model.

• **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, a Structural Model, and a Task Model.

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, outputing: a Knowledge Model, a Behavior Model, a Responsibility Model, and a Goal Model.

• **Architectural Design**: In this stage, we decide whether or not the system will be designed following a multiagent architecture. If an MAS is designed, the entities of the architecture are also defined. The generated models are: an Agent Model, an Object Model, and Responsibility Models.

• **Society Design**: The Multiagent Architecture Design can result in an agent society, in which the system is designed as a set of agents embedded in a social structure. Several models are generated in this stage: a Social Agent, a Role, a Relationship, and a Social Commitment Policy.

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

Although the methodological process is top-down, this methodology follows a bottom-up process to build the MAS architecture (Figure 3). 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 or building block), 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.

Each stage and phase of the methodology matches a level of the pyramid shown in Figure 3. The bottom level, Conceptual Level, contains atomic elements (concepts, associations, tasks, methods and external entities) identified by the analysis phase. The Component level (knowledge, behaviors, responsibilities and goals) contains aggregated elements identified by the synthesis stage, the Entity Level contains agents, objects and interactions identified by the multiagent architectural design stage, and, finally, the Social Level describes social agents, roles, relationships, and social commitment policies, representative elements of the society.

4.1 Conceptualization stage

The elicited requirements are analyzed using the Set Theory Based Conceptual Model (SETCM) [37,38], an analysis method that was defined to combine a formal foundation with a pragmatic approach. This analysis method is design-
independent: it uses a terminology other than design language to provide a real comprehension of the problem under analysis. It 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 [38–40].

The modeling elements of SETCM have been defined carefully, choosing common elements existing in other approaches, eluding design-specific terms and incorporating new elements where needed. These elements have been defined using set theory terminology, which is the basis of mathematics.

SETCM divides a conceptual model into two models. The first is the Structural Model, which represents the structure of some domain (elements and relationships between them) and the states that can occur within this domain. The second is the Task Model, representing problem solving in the domain. Problem solving is expressed as the process of making a series of changes to the system state, from the initial state (the problem) to the final state (the solution).

The Structural Model is composed of seven different sets (Table 1) and the Task Model consists of two main sets, whose definition depends on the use of two predefined sets: operators and query functions (Table 2).

SETCM has a formal basis, involving the formalization of all the modeling primitives. These primitives are formalized using the main elements of Cantor’s naïve Set Theory, but defining a rigid modeling structure that eludes the contradictions of this theory. SETCM has a formal basis, which means that well-founded analysis decisions can be made, because the implication of each decision can be ascertained with exactitude.

SETCM solves pragmatic issues by defining two notations for the analyst: an easily readable textual notation that can represent all the SETCM modeling primitives and a UML-based graphical notation that will be used in this paper. As mentioned earlier, SONIA’s Initial Structural Model and the Initial Task Model are built using SETCM.

### 4.2 Extended analysis stage

Having conceptualized the problem, the models built are refined and expanded to capture the system environment and external entities. When a problem is analyzed for modeling in a computer, the environment with which it is related becomes part of the problem or at least of its relationships. Even though analysis covers the environment, the analysis models will still depend on the problem viewpoint and analysis will remain independent of the design paradigm. The Extended Analysis Stage produces the following models:

- An Environment Model, which defines the external system entities and system interactions with these entities.
- A Structural Model, which can extend the system knowledge with knowledge that the external entities supply to the system.
- A Task Model, which can extend the tasks performed by the system with any tasks required to interact with external entities.

At the end of this analysis stage, it can be decided, initially, that if the system does not feel and act in the environment or is not reactive to environmental changes, it should not be modeled as an MAS.

### 4.3 Synthesis stage

The step from analysis models to architectural design models is usually a traumatic process because they are too far apart. The adoption of design paradigm-dependent analysis solutions is clearly conditioned by the chosen design model. Therefore, a design paradigm-independent and an intermediate analysis phase are proposed to smooth the transition to the system architectural design (Figure 4).

This approach is based on synthesizing information contained in the analysis models as higher level structures related to the reference paradigm. Synthesis is the first design phase, whereby the viewpoint switches from the domain (analysis) to the solution (design). Consequently, it should ease the identification and formalization of computable structures that are coherent with agent orientation from the analysis models.

The Synthesis stage provides for the later component-driven identification of agents by grouping elements of the Structural Model and Task Model that make up concepts characteristic of agents, such as knowledge, behaviors and responsibilities. It produces the following models: a Knowledge Model, which identifies the knowledge blocks inherent to the problem by grouping concepts and associations from the Structural Model; a Behavior Model, produced by grouping tasks, subtasks and methods from the Task Model; a

### Table 1 Sents of the SETCM structural model

<table>
<thead>
<tr>
<th>Set</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Set of values. There are values in every state. Types can be either basic or derived</td>
<td>Assimilative technology (derived, set of strings)</td>
</tr>
<tr>
<td>Concept</td>
<td>Set of instances. The instances of a concept conform to its invariant in every state</td>
<td>Person, Aptitude, Recommendation</td>
</tr>
<tr>
<td>Attribute</td>
<td>Correspondence between a concept and a type. It assigns type value(s) to each instance of the concept</td>
<td>Age (for concept Person)</td>
</tr>
<tr>
<td>Association</td>
<td>Correspondence between concepts, types or even other associations. This allows for higher order associations</td>
<td>Decision (between several Agitudes and a Recommendation)</td>
</tr>
<tr>
<td>Attribute</td>
<td>Correspondence between an association and a type.</td>
<td>Reliability (for association Decision)</td>
</tr>
<tr>
<td>Classification</td>
<td>Relationship between a general concept (superconcept) and more specific concepts (subconcepts)</td>
<td>Person can be classified as Male or Female</td>
</tr>
<tr>
<td>Association</td>
<td>Relationship between a general association and more specific associations</td>
<td>Decision can be Intermediate or Final</td>
</tr>
</tbody>
</table>

### Table 2 Sents of the SETCM task model

<table>
<thead>
<tr>
<th>Set</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Correspondence in the state universe. It relates initial states (problems) with final states (solutions). It is declarative. It specifies inputs, preconditions, postconditions and outputs</td>
<td>Analyze user</td>
</tr>
<tr>
<td>Conceptualization Models</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task method</td>
<td>Definition of the decomposition of a task into subtasks and the specification of how to control and combine the performance of subtasks</td>
<td>Analyze User = Take Survey = Analyze Aptitude</td>
</tr>
</tbody>
</table>

![Figure 4 Transition from analysis models to design models](image-url)
Responsibility Model, output by establishing the relationships between knowledge blocks and behaviors, and a Goal Model, which represents the main objectives of the system.

### 4.3.1 Knowledge Model

The **Knowledge Model** can identify the knowledge blocks by grouping Structural Model concepts and associations. The knowledge blocks will be used internally or shared by the agents. It should be noted that the structural model allows for the representation of very complex domain structures, including elements that could represent not only information, but also knowledge (i.e. rules).

These groupings are identified on the basis of the concepts and associations of which they are composed, and:

- Are strongly related to each other. The groupings are internally highly cohesive.
- Have little relationship to the other concepts and associations (low-coupled grouping).
- Are used to perform the same tasks.

A new technique, based on Kelly’s Trait Analysis [41], was used to get the first version of the Knowledge Model [36]. This technique identifies groupings of concepts and associations that have the first two properties of a knowledge block: high cohesion and low coupling. To systematize the application of this technique, the conceptual diagram of the Structural Model is first transformed automatically into a directed graph. Then a 2D table is built that stores the connectivity (no. of arcs or connections) between each pair of nodes in the graph. Finally, the new Trait Analysis-based technique is applied using the connectivity between two graph groupings as a measure of comparison.

The resulting groupings of the first version of the model will only satisfy the first two conditions. The final version of the model, which is output when the responsibilities between knowledge blocks and behaviors (Responsibility Model) are established, will meet all the conditions.

The Knowledge Model describes both the knowledge blocks and the relations between knowledge blocks. The knowledge blocks are represented as UML classes with the stereotype «knowledge». The result of the model is a **Behavior Diagram** that describes the organization of behaviors (identified behaviors and time dependencies between them).

### 4.3.3 Responsibility Model

The **Responsibility Model** is output by relating knowledge blocks to behaviors. This model is essential for identifying agents and environment objects. A basic activity during model design is to refine the Knowledge and Behavior Models to meet all their conditions.

In this model, we describe the responsibilities between knowledge and behavior using a **Responsibility Diagram**, which illustrates the relationships of knowledge use by behaviors.

### 4.3.3 Goal Model

The **Goal Model** is composed of the system objectives. These goals are logical conditions imposed on the state of knowledge and are identified from Behavior Model task postconditions. The ultimate aim of this model is to be able to identify agent goals. The agent will execute behaviors to achieve its goals.

### 4.4 Architectural design stage

The purpose of the second stage of **Multiagent Architecture Design** is to define the architectural elements by means of the following models: an **Agent Model**, which identifies and defines, from the Knowledge, Behavior, Responsibility and Goal Models, what elements should be designed as autonomous agents; an **Object Model**, which identifies and defines, from the Knowledge and Responsibility Models, what passive elements there are in the environment; and an **Interaction Model**, which identifies and defines the relationships between the system and agents and between agents and objects.

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 be considered as an autonomous agent, it should have a behavior and the right knowledge blocks 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. If you select this paradigm, you can reuse Object and Interaction Models built in the multiagent architectural design stage. The objects identified in the Object Model will become OO design objects and the interactions identified in the Interaction Model will be method invocations in the OO design.

Another possibility would be to design the system as a knowledge-based system. In this case, the designer can reuse the knowledge blocks, behaviors and responsibilities output in the synthesis stage.
4.4.1 Object Model
The Object Model identifies and defines what passive elements are part of the environment. These objects are knowledge identified during the Synthesis stage from the Responsibility, Knowledge and Behavior Models.

The main characteristic feature of an environment object distinguishing it from other system knowledge is that the knowledge of this object is the responsibility of more than one behavior or, in other words, it is shared by several behaviors.

Access to objects will be divided by levels, and the knowledge blocks that are accessed by the same behavior tasks will be grouped at the same level. The levels are a way of structuring the knowledge contained in the object in a blackboard-like fashion.

4.4.2 Agent Model
Research into agent architectures has engendered three different and well-defined types of architectures for the design of autonomous agents. The deliberative or symbolic architecture considers agents as knowledge-based systems with learning-planning paradigm in AI [42]. In the reactive or behavior-based architecture, an agent is modeled according to a group of behavioral, level-layered tasks, which must be carried out by the agent [43]. The third is the hybrid architecture, following the BDI (Belief-Desire-Intention) paradigm [44]. Our agent architecture conforms to a BDI architecture, because it combines proactive and reactive behavior.

The Agent Model identifies and defines what entities should be designed as autonomous agents (active entities of the environment). Agents are identified from the knowledge gathered in the Synthesis phase from the Responsibility, Knowledge, Behavior, and Goal Models. An agent is an environment-sensitive entity (which senses and acts in the environment), which 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.
- **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. These goals are logical conditions on the state of the environment objects or the state of their internal knowledge.
- **Utilities** are triggers that activate the agent. The agent will assess the execution of some of its behaviors if their utilities are met. These 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.

The behaviors, goals and utilities characterize the role that the agent plays in the system.

4.4.3 Interaction Model
The Interaction Model identifies and defines what relationships there are in the system between 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 [45]. However, agents do not participate in isolated message exchanges, they enter into conversations, i.e. coherent message sequences designed to perform specific tasks that require coordination, such as negotiations or agreements. This exchange sequence may emerge spontaneously or have been agreed upon beforehand and specified a priori by means of an interaction protocol (IP). In the case of a reductionist MAS system, the interaction protocol is designed at the same time as the actual agent. In the case of a constructivist MAS system, the interaction protocols are located in a library and are accessed by the agents at interaction time. This a priori and constructivist approach necessarily depends on the provision of a framework to support the modeling of interactions between agents that considers all the stages of a protocol engineering process, i.e. the design, specification, validation, implementation and management of IPs considered as resources. We have developed such a supporting framework for SONIA, called ACSF [46] (Agent Conversation Specification Framework). ACSF is divided into three views. The modeling view allows visual IP design. The specification view automatically out-puts, the syntactic specification of the IPs from the design in a declarative-type language called ACSL (Agent Conversation Specification Language), which improves IP publication, localization and machine learning by agents. Finally, the implementation view provides an operational semantics for the ACSL language. This semantics allows protocol property verification and eases automatic code generation from the ACSL specification for the purpose of simulating code execution at design time, as well as improving and assuring correct IP compliance at run time.
- **Agent-object relations** are implemented through message passing and occur when an agent accesses an object level, either through a sensor or an actuator.

The methodology accounts for the two communication types: asynchronous communication, using environment objects (or blackboards) to subscribe to events of interest to the agent; and synchronous communication, through protocols contained in the Interaction Model.

4.5 Design of the agent society
Agents participate in a society to achieve their own goals using social resources. The society defines some rules (termed social commitment policies) for both joining the society and acting within the society, and any agent wanting to participate in the society should abide by these rules.

In [47] we introduced a society conceptual metamodel,
which can be used to design the society and to define how this society would be organized. The key concepts of the proposed metamodel are the social agents, the roles assumed by these agents, the relationships between these roles, the social commitment policies that govern these relationships and the social structure that arises as a result:

- **Social agents**, or active entities of the society. We propose a taxonomy characterizing all the agent types that would take part in a society [47]: Member Agents are specialized in performing a given social task in the context of an organization (i.e. an MAS inside the society). Client Agents are individualistic entities and can choose to cooperate, or not cooperate if they do not see any direct benefit deriving from such cooperation. Socially Responsible Agents provide knowledge, services and resources selectively on the basis of an individual/social weighting in their own interests and for the benefit of society as a whole, retaining their local autonomy and taking advantage of interactions with other agents and with organizations. Interface Agents represent an organization (i.e. an MAS inside the society) before the society. Their responsibilities include handling any access and service requests from client or socially responsible agents, as well as from other interface agents, addressed to the organization they represent.

- **Roles**. Each agent has its role or well-defined position within the society. This role motivates and structures all interactions between and assigns abilities to agents. A role determines the following competence areas for the agent that plays it: services provided, responsibilities, interaction patterns and protocols, and information requested and provided for agents to fulfill their responsibilities.

- **(Social) relationships**. A feasible way to characterize a society is by identifying the potential social relationships that could take place among its members and, more specifically, among the different identified roles. As the society is open, these are dynamic relationships. The agents taking part in a specific relationship can behave in different ways (benevolently, individualistically or socially cooperatively), which leads to different kinds of relationships (control, peer-to-peer, benevolent, dependence or property relationships).

- **The social commitment policies**, which regulate the potential social relations between the society’s actors and, specifically, between all the identified roles. These policies can be implemented using blackboards that describe specific agents that are responsible for reporting these laws to agents and taking action when they are disobeyed.

We present in [48] an innovative role-based organizational model for formal agent society design and specification. The concept of social commitment policies (norms, rules, social laws, etc.), which govern various aspects of group behavior, including penalties for individual acts in the context of group activities that violate the established social commitments, is at the heart of this model. The different types of social commitment policies are classified and formalized, and the interest of their application not only as an organizational abstraction but also as a mechanism for specifying the expected behavior of the agent society is discussed. This formalization will allow agents to reason deliberatively about the existing social norms, the responsibilities proper to their position and the consequences of violating these responsibilities.

5. **CASE STUDY**

An application of this methodology was ALBOR (Barrier-Free Computer Access) [38,49]. ALBOR is a real project funded by the UPM (Universidad Politécnica de Madrid) and IMSERSO (Spanish Institute of Migrations and Social Services). 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 the four stages described below:

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

![Figure 5 Conceptual diagram of ALBOR Structural Model](image-url)
• **Survey Taking:** the user is asked a series of questions, which will depend on responses to questions already answered and will be confined to the questions strictly necessary for the evaluation of the person in question.

• **Result Evaluation:** after collecting all questionnaire data, an evaluation report is sent to the user. This evaluation contains several recommendations (sorted by priority) for the user to decide which is best suited for her/him.

ALBOR is conceived to operate in a network of centers located in different countries so that each country can customize the questionnaires and assessment rules according to the recommendations in force in the respective country and the disabilities in question. Therefore, ALBOR should make provision for replicating the intelligent system and updating knowledge independently in each copy, and should include mechanisms for knowledge consistency maintenance in the copies.

In the following, the phases and stages of the SONIA methodology are applied to the development of the intelligent system within the ALBOR project. To simplify the case study, the network-inherent features, such as copying, knowledge updating and knowledge consistency maintenance, will not be considered. Also for simplicity’s sake, only the methodology for identifying the multiagent architecture will be applied.

### 5.1 Analysis of the ALBOR system

The first stage of the analysis was to conceptualize the ALBOR system using the SETCM method. As a result, a series of concepts, associations and attributes that give a conceptual view of the problem (Initial Structural Model) and a series of tasks and problem-solving methods that give a procedural view of the problem (Initial Task Model) were identified.

Then the environment was analyzed to identify the external entities and the interactions between these external entities and the ALBOR system. New concepts and associations emerged from the knowledge of the external entities and were added to the Structural Model (Figure 5) and used in the Task Model to specify task preconditions and postconditions (Figure 6).

As ALBOR handles a lot of information, the conceptual diagram is quite large, and it has, therefore, been simplified by omitting the attributes of both concepts and associations.

The graphical notation used in Figure 5 is an extension of UML. Types, concepts and associations are represented as stereotyped classes; and directed lines link the participants in the association: an arrow going to an association represents one of the source participants and an arrow going from the association represents the target participant.

With respect to the task model, in some problems (as in the ALBOR system), there is only one way of solving each task, that is, there is only one method per task for such problems. Hence, the graphical notation of the task/method diagram used in Figure 6 represents each task/method group in a UML class, and the relationship between a task and the subtasks is represented by UML aggregations.

At this point, a decision can be made about whether or not to continue with the multiagent approach. As the Structural and Task models are highly complex and dependencies have been established between tasks and elements of the environment through task pre- and postconditions (reactive behavior), it should be modeled as an MAS.
5.2 Synthesis of the ALBOR System

The groupings of concepts/associations output (by applying the technique based on Kelly’s Trait Analysis) have been marked on the graph in Figure 7, and we find that not all the concepts and associations have been able to be grouped in this first step. No distinction is made between concepts and associations in the graph after conversion of the conceptual diagram.

These groupings are then inspected to find out whether or not they are valid, for which purpose constraints are applied to the relations between groupings.

Table 3 shows the first version of the Knowledge Model that contains the groupings (of concepts and relations) identified above and their relations. The concepts and relations added to be able to satisfy the constraints on the relations between groupings are specified in italics.

One grouping for each first-level task, including the grouping of its subtasks, was generated to get the first version of the Behavior Model. The dependencies between groupings are identified from the task preconditions and post-conditions. Table 4 shows the content of the groupings and the dependencies with other groupings.

To get the Responsibility Model, the first versions of Knowledge and Behavior Models were modified to output their final versions and assure that both knowledge and the identified behaviors satisfied the required properties.

The Knowledge Model was modified from use relations of concepts/associations in tasks/subtasks. Some inclusion rules...
were defined [36], which determined when to include a concept/association in an existing knowledge block or when to group a series of concepts/associations as a new knowledge block.

Figure 8 shows how the concept ‘User’ and its associations ‘User.HasCurrentQuestion’, ‘User.HasResponses’, ‘User.HasAptitudes’ and ‘User.HasRecommendations’ were grouped to produce the knowledge block ‘Users’ and add the concept ‘Medium’ to the knowledge block ‘Medium’. The model was completed by identifying relations between knowledge blocks from use relations between the elements of the Structural and Task Models.

The modification of the Behavior Model focused on the identification of groupings or divisions of identified behaviors. The goal of this stage is to refine the groupings output in earlier stages, including the information gathered from the use relations of concepts/associations in tasks and subtasks. To ease this identification, a set of grouping/division rules [36] are used that can evaluate whether several behaviors should be grouped as one or one behavior be divided into several.

As a result of applying these rules, we decided to divide the ‘AnalyzeUser’ behavior into two behaviors, namely, ‘TakeSurvey’ and ‘EvaluateAptitude’, in ALBOR. On the other hand, there were no behavior groupings. All that remained to complete this model was to modify the dependencies between the new behaviors. Figure 9 shows the new behaviors and the dependencies between them.

Having output the final versions of the Knowledge and Behavior Models, the relations between the two could be set out in the Responsibility Diagram (see Figure 10). To simplify this diagram, the knowledge and behavior content, which can be looked up in the Knowledge and Behavior Models, has been omitted.

All that remains to complete the synthesis stage is to identify the system goals. The objectives are taken from the post-conditions of the tasks that make up the behaviors. Table 5 shows the goals associated with the identified behaviors. For example, a single goal (new Users.recommendations) has been identified for the EvaluateAptitude behavior, whose meaning is to get new recommendations for a user. We find that this goal is defined on an environment object, which means that the agent pursuing this goal will be driven more by the environment than its internal behavior.

5.3 Architectural design of the ALBOR system

The environment agents and objects were identified during ALBOR architectural design. 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 (white box in Figure 11).

The candidate agents were also identified from responsibilities between knowledge blocks and behaviors. One candidate agent will be output for each knowledge block that is the responsibility of a sole behavior (relation 1:1).

In the case of ALBOR, three candidate agents were identified: ‘Survey-Taker’ agent (‘TakeSurvey’ behavior and ‘Questionnaires’ knowledge block), ‘Decision-Maker’ agent (‘EvaluateAptitude’ behavior and ‘Rules’ knowledge block) and ‘Advisor’ agent (‘ShowReport’ behavior and ‘Recommendations’ knowledge block) (Figure 11). The ‘InitSession’ and ‘IdentifyUser’ behaviors were not assigned, because they do not have any knowledge proper and, therefore, do not qualify as candidate agents.

Not all candidate agents will be converted to agents. This will be confined to agents that meet all the requirements for becoming an autonomous agent, i.e. agents that have at least one defined goal and utility, and sense and act in the environment (have at least one sensor and actuator).

For example, the ‘Decision-Maker’ agent has one goal and one utility and senses and acts on the ‘Users’ object, whereas the other two agents sense and act on all three objects: “Users”, “External” and “Media” objects. The three candidate agents identified earlier meet these two conditions and qualify as autonomous agents.

Finally, all that remained was to add the discarded behaviors and knowledge to the existing agents, for which purpose the Behavior Diagram of the Behavior Model was analyzed. As a result, the behaviors (‘InitSession’ and ‘IdentifyUser’), which had no assigned knowledge block, were added to the ‘Take-Survey’ agent.

Figure 9 Behavior Diagram (dependencies between behaviors of Behavior Model)

Figure 10 Responsibility Diagram (responsibilities between knowledge blocks and behaviors)

Figure 11 Identification of objects and candidate agents
Figure 12 shows the resulting multi-agent architecture of the ALBOR system, which illustrates the identified agents and objects together with their components and interactions.

6. CONCLUSIONS

Agent-oriented software engineering is at a stage where, if it is to firmly establish itself as a genuine engineering discipline, it needs an agent modeling language and a consolidated agent development process, as object-oriented software has. Although there is still a lot of work to do, this should be a solution in the near future if we want the agent paradigm to be incorporated into routine industrial software development processes.

This paper aims to contribute to the methodological issue of agent-based development by setting out what the basic requirements for an AOSE methodology should be and defining a new methodology that meets these requirements. For this purpose, we have listed what features an agent-oriented development methodology should have and detailed which of these features are missing from the most commonly referenced methodologies used within the agent paradigm. Finally, 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 society development.

Although we believe that the SONIA methodology is conceptually well grounded, it does have some shortcomings, including the fact that there is no CASE tool for generating models, it does not account for other agent architectures apart from BDI, it is difficult to apply in open and heterogeneous environments, and does not provide for agent reuse, etc.

We are now working on the development of an appropriate case tool that supports the automation of methodological processes and generates skeleton code for an agent language, like JACK. We also intend to apply the SONIA methodology to other application domains, like network management or workflow systems.

ACKNOWLEDGEMENTS

We would like to thank the anonymous referees for their valuable comments that have been extremely useful for improving the paper.

REFERENCES


The FIPA Agent UML Web Site http://www.auml.org/


Wavish, P and Graham, M A situated action approach to implementing characters in computer games, Int. J. of Applied Artificial Intelligence, 10(1) (1996) 53–73


Alonso, F, Frutos, S, Fuertes, J L, Martinez, L A, and Montes, C ALBOR. An Internet-Based Advisory KBS with a Multi-Agent Architecture, Int. Conf. on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2001), L’Aquila, Italy (2001) 1–6


Austin, J L How to Do Things with Words, Harvard University Press, Cambridge, MA (1962)

