Nemo: an Agent-Oriented Software Engineering Methodology

Marc-Philippe Huget
Agent ART Group
Department of Computer Science
University of Liverpool
Liverpool L69 7ZF
United Kingdom
M.P.Huget@csc.liv.ac.uk

Abstract

The development of multiagent systems is increasing significantly. Thus, multiagent system designers need methodologies and tools to help them. Recent years have seen an incredible development of agent-oriented software engineering methodologies trying to cover all the range of features that agents and agent-based systems encompass. These methodologies are either based on object-oriented methodologies or specific to agent theory. The Nemo methodology is a new agent-oriented methodology. Its main advantage is to tackle notions such as mobility, security or open large scale multiagent systems. These notions are more or less not yet considered in agent-oriented methodologies. This paper presents the Nemo methodology and its eight models describing organizations, plans, roles, interactions, knowledge or agents.

1. Introduction

Recently, the development of multiagent systems is increasing significantly, particularly in the domain of electronic commerce and business [32]. At the same time, designing multiagent systems becomes more and more complex. It is no longer possible to consider designing agents and agent-based systems without some help. This help is provided by methodologies and tools.

A commonly-held idea is to consider agents and objects as equivalent notions. This view is too restrictive to show all the richness of agents. As Odell [36] and Wooldridge [45] note it, even if agents and objects have some points in common, many differences exist between these two approaches. The main differences are autonomy and interaction. Actually, agents are autonomous, reactive, proactive and intelligent. Agents are capable of initiating action independent of any other entity. Autonomy for agents means that agents are able to react to events occurring in the environment. Proactive agents will actually poll the environment for events and other messages to determine what action they should take. They are not triggered by other agents or by humans. Finally, interaction between agents is richer than the one in object-oriented systems and also more abstract through high-level messages. Interaction in object-oriented systems corresponds to method calls. Interaction between agents is completely different. Messages between objects are simpler since it is only possible to invoke methods and give parameters. These parameters are fixed and if designers want different kinds of parameters, they have to provide as much methods as they have different kinds. Agents use agent communication languages such as KQML [14] or FIPA's ACL [15] to exchange messages. The aim of agent communication languages is to provide a precise syntax and semantics for interaction between agents.

Several other areas where agents and objects differ include scheduling, learning, adaptivity, multiple and dynamic classification or the emergence. As soon as agents have goals, they derive plans to fulfill them. By learning and adaptivity, we mean that agents have the ability to modify their behaviors given events occurring in the environment, or to acquire new behaviors based on their observation of other agents or of the environment. Multiple and dynamic classification refer to the ability for agents to have different roles during an execution and to move from one role to another one during it. Emergence within multiagent systems means that an overall behavior emerges from the interactions between agents [35]. We let readers consult [36] [46] for further details and other differences.

Due to the richness and the complexity of agents and agent-based systems, existing software development techniques (for example, object-oriented analysis and design [31]) are unsuitable to develop agent and agent-based systems. Two main directions have been considered by mul-

tiagent system designers: extending software engineering (or knowledge engineering) methodologies or defining specific agent methodologies. Recent years have seen an incredible development of agent-oriented software engineering methodologies trying to cover all the range of features that agents and agent-based systems encompass. Readers are urge to read the helpful survey from Iglesias et al. to this purpose [25]. Even if it does not contain most recent methodologies, it presents a good overview of agent-oriented methodologies. Most cited or recent methodologies are MESSAGE [5], Tropos [33], PASSI [4], MASSIVE [28], Gaia [46], MaSE [8], DESIRE [3], AALAADIN [12], CASSIOPEIA [6], VOWELS [9], Prometheus [39], MAS-CommonKADS [24] and CoMoMAS [17].

There are two options when designers do not find their needs in current methodologies: (1) update a methodology or (2) define a new one. The former is the fastest one since designers "just" have to enhance a methodology. However, this approach is painful if (1) the new needs are difficult to include in the methodology and (2) the methodology allows few improvements due to its lack of flexibility. Updating a methodology is the approach followed by Juan et al. in [26] where they extend Gaia.

This paper addresses the latter option: the definition of a new methodology called Nemo. The main benefit to define a new methodology is that designers are sure to find a methodology perfectly tailored to their needs. The work is certainly more considerable than updating a methodology but it is easier to keep it easily updated.

The Nemo methodology has two main features that distinguishes it from other ones: (1) mobility and security management and (2) open large scale multiagent system management. Actually, most of the methodologies aforementioned only consider reactive and cognitive agents. The Nemo methodology focuses on the analysis and design stages and provides eight models representing organization, interaction, agents or tasks.

The Nemo methodology addresses a wide range of multiagent systems. Multiagent systems can be open (with a large number of agents) or closed, agents can be reactive, cognitive or rational. They can be homogeneous or heterogeneous. Finally, agents can be mobile.

The Nemo methodology is a new methodology and is still ongoing research. As a consequence models should be refined in a near future. The paper focuses on the different models and not on a particular application of this methodology.

The paper is organized as follows. Section 2 describes an overview of the Nemo methodology. We present briefly the meaning of the eight models. Section 3 presents the models in detail. Section 4 compares this methodology with others. Section 5 concludes the paper and gives future directions.

2. The Nemo Methodology

The Nemo methodology is an agent-oriented software engineering methodology. It addresses the analysis and design stages. The main benefits of Nemo in comparison with other methodologies is to take into consideration the agent mobility and the security attached to this mobility. Moreover, the notion of open large scale multiagent systems is addressed in Nemo.

The development of multiagent systems is performed through eight models (mobility and security are not defined as specific models but are included in the other models):

Organization model This model shows the organizational structures within multiagent systems. Organizations in the context of multiagent systems correspond to groups having a particular structure and specific rules. Since organizations can be open large scale multiagent systems, several new attributes are included with the usual ones (agents' roles, services and roles' relationships), such as roles' cardinality, procedures for entering and leaving, norms, functioning costs, and procedures for splitting or merging organizations or organization knowledge.

Plan model This model depicts the different plans used within multiagent systems. A plan is composed of tasks which correspond to actions done by agents. These tasks are organized as trees. The plan model contains conditions required to perform the plan, agents involved in this plan and plan substitutes. A task is described as a procedure to complete it as well as the conditions under which it is performed.

Interaction model This model depicts the interaction between agents. These interactions are performed through protocols. The interaction model is decomposed into two parts. The requirement analysis document of the protocol is done in the first part. The second part describes a graphical representation of the protocol. It represents the sequence of messages for this interaction. Nemo uses Agent UML sequence diagrams [37] to this purpose. Security is a major problem in the context of interaction. This problem is addressed in the interaction model.

Environment model The environment model is particularly important for reactive agents which evolve in an environment. The environment model shows the environment structure and the different entities (agents, resources) belonging to it. If multiagent systems are open, it is possible to add some rules to prevent agents from consuming all the resources. A second use of these rules is to secure the multiagent systems from malicious agents.

Role model This model depicts the different roles, their relationships and how roles are organized into groups. These groups are different from the ones defined in the organization model. They are only used to represent the permissions, authorizations on knowledge and actions. A role belongs to a group and as a consequence has some permissions. The description of a role follows Gaia's approach [46].

Agent model This model shows the entities (the agents, the objects (usually the resources)) and the relationships between entities. This model is based on Agent UML class diagrams [22] and is extended with notions such as mobility and knowledge privacy.

Knowledge model Knowledge is one of the main differences between agents and objects. Agents encompass beliefs, desires, intentions and knowledge about their users, other agents and the environment. The knowledge model is similar to the one found in [7] except the notion of data privacy.

Deployment model Even if it is stated in introduction that the Nemo methodology focuses on the analysis and design stages, it is important to propose a deployment model at this level of design like in UML [2]. It is particularly important to have this model in the context of mobility and security. This deployment model describes how entities (agents, resources and software) are spread on the different machines. This deployment model is based on the UML deployment diagrams and is extended to cover the notion of mobility and security management.

3. Nemo's Models

The Nemo methodology is composed of eight models. Developing a multiagent system with the Nemo methodology consists in filling these models or a subset of these models. The design can be iterative and the models can be fulfilled in any orders. For instance, if the organization is the main aspect in a multiagent system, designers should begin with this one. The Nemo methodology follows the idea of abstraction in UML: all the models are not required and even in a model, it is not required to fulfill all the information. It depends on the targeted multiagent system. The security in each model does not have to be provided if it has no sense for the developed multiagent system.

3.1. Organization Model

Agents within multiagent systems are gathered into groups similar to human ones such as manufacturing cells, organizations, or markets [16]. We consider organizations

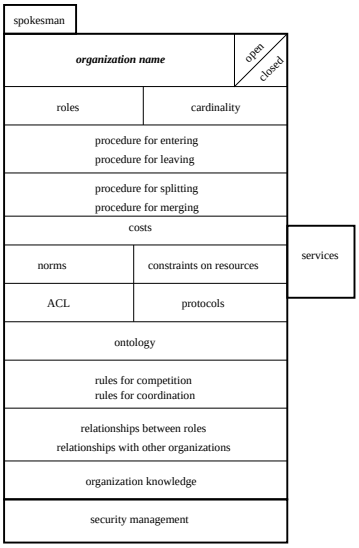

Figure 1. Organization Model Schema

to be a subtype of group. Here, an organization is defined as a group whose roles and interactions are typically expected to be relatively stable and change slowly over time [40].

In the MESSAGE methodology, organizations are described through organization diagrams [5]. These diagrams contain information about the different roles involved in this organization and the cardinality for each roles. This approach is too restrictive in comparison with the richness of multiagent societies or human societies. To this purpose, we follow an approach related to institutions and electronic institutions [10]. Nevertheless, we extend this idea with new elements such as mobile agent management and security management to name a few.

The organization model is rendered as a schema as shown on Figure 1.

An organization is distinguished by a name (the *organization name* compartment). The openness of a system is given by the keyword *open* else the system is closed and the keyword *closed* is used. An organization contains a set of roles. These roles represent the agents' behavior in the system. To tackle large multiagent systems, we add the *cardinality* for each roles. This piece of information constrains the number of agents per roles in the system. It prevents some problems like the lack of resources. For instance, in auctions, the cardinality is the following one: one and only one agent of role *auctioneer* and at least two agents of role *participant*.

Entering or leaving multiagent systems implies the addition of procedures to control the flow of agents. Entering procedures are defined as protocols, algorithms or free-format text. Other information can be inserted in this procedure such as the benefit of this new agent to the sys-

tem. Such verification is performed through utility functions. Examples of such utility functions are in [17]. Only agents which can improve the utility of the system could enter the system. An example could be: *"All agents that want to enter in the system have to request their entrance to the spokesman*".

Leaving procedures are protocols, algorithms or free-format text. Such an example of leaving procedure is the following one: "*All agents that want to leave the system have to inform the directory facilitator*". The directory facilitator agent stores the agent's address.

Procedures for splitting refer to large scale multiagent systems. When too many agents have the same role and the redundancy is too important, it is better to split the organization into several parts. The splitting procedures compute when the separation may occur. The procedures are algorithms or free-format text.

Procedures for merging are the counterpart of the splitting procedures. They are used to check if some organizations can be united. Merging organizations is meaningful when several organizations depend frequently on each other for their tasks and services. It is possible to merge if the cost associated to the merge minus the cost of redundancy is less than the cost of dependency between organizations. Such procedures may use utility functions.

Like human organizations, multiagent system organizations represent a *cost* of functioning. This cost is computed given the number of agents in the system and other parameters such as the level of resources.

Large scale multiagent systems need rules to prevent abnormal situations or to ban agents behaving improperly relative to the organization goal. These rules are called *norms*. Organizations and especially human organizations are constrained by social laws [19]. These social laws are defined by an authority and social laws are obeyed since they are agreed upon. The same notion is present in multiagent systems except that norms deal more with constraining agents' behaviors [30]. Norms are also seen as a specific means to fasten the task completion. Norms are particularly present in electronic institutions [10] and especially in the example of Fishmarket where auction winners have to pay the price of the items when requested. Added to norms, there are penalties which prevent agents not to follow norms. Norms and penalties can be defined formally or informally. Garcia et al. [10] choose to represent norms and penalties formally. An example of norm is the following one: "*if an agent gets an item at auction, it has to pay the requested price*".

Another feature refering to large scale multiagent systems is *constraints on resources*. This piece of information avoids agents consuming all the resources. Some punishment can be associated to this piece of information.

Like human organizations, this organization model defines the language in use within it (the compartment *ACL*). It means that agents preferably use this language. However, it is not a requirement for agents and heterogeneous societies are possible when ontologies are applied. An ontology defines the meaning of terms and concepts and describes the relationships between the elements [11]. Ontologies are practical to deal with heterogeneous agents. Then, agents which do not share the same vocabulary can interact each other if they are able to know how to translate words from one language to another one. This feature is interesting in a context of interoperability.

Entering the system, leaving it, or regulating it needs protocols. These protocols have to be distinguished from those used by agents and specific to roles. These protocols are defined through the interaction model (see Section 3.3).

Competition rules and *coordination rules* refer to dependency relationships between organizations. They are used to define the general policy of the organization in the context of task advertisement. Once again these rules are related to utility functions. For instance, if the competition against another organizations for tasks or services is a good thing for the organization in term of market share, the organization applies for this task. If the coordination with other organizations is better to cut cost off, then the organization enters in a context of coordination with other organizations. Of course, if other organizations involved in coordination refuse this coordination, then the organization can enter in a context of competition.

Roles within organizations are organized according to relationships. Here are several relationships:

1. unilateral dependency where an organization depends on the others. This relationship is not reversible. It means that if A has a dependency over B, B has no dependency over A. The two way relationship of dependency is the following one.

2. mutual dependency where the dependency is a two way relationship. If A has a dependency over B, B has also a dependency over A. It is the case if A needs a result that B can provide to it but B needs information that A has.

3. acquaintance corresponds to a relationship where two organizations know each other. This relationship is used when organizations are not in a context of cooperation or coordination.

4. the relationship of power is essentially used in hierarchies to represent that an organization higher in the hierarchy gives order to organizations lower in the hierarchy.

AOR presents other relationships related to organizational relationships [43].

Human societies contain a common knowledge and materials to people belonging to them. This notion is also applied to multiagent systems. The organization knowledge contains information such as mutual beliefs and intentions [44]. Knowledge is then provided to all agents in the organization.

Two pieces of information are provided outside the organization frame: services and spokesman. We place them outside to underline that they represent the entry points to agents outside this organization. *Services* are the services performed by this organization. They are defined as a free-format text. Behind services, there are tasks defined in the plan model (see Section 3.2). *Spokesman* is the agent interfacing this organization to other agents and organizations. When an agent wants to enter this organization, it asks this agent. When an agent or an organization asks for a service, it asks the spokesman. The spokesman is a specific role attached to the organization.

We add a new compartment at the bottom of the diagram to deal with security management. This compartment contains a set of information such as security rules. These rules can refer to what the system has to do when new agents come into the system or ask the system. Rules can also concern what the system has to do in case of agent attacks. For instance, a rule might be *if an agent consumes abnormally all the resources, the system removes it from the system and shuts all the communication as long as the level of resources is too low*.

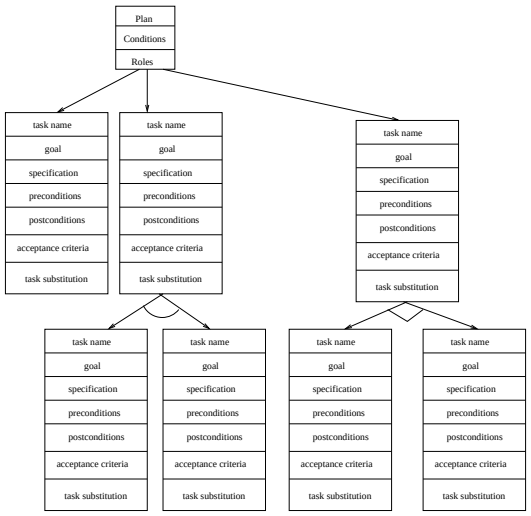

Figure 2. Plan Model Schema

3.2. Plan Model

Cognitive or rational agents define plans to fulfil their goals. These plans are composed of atomic operations called tasks. We refer to atomic operations to underline that no further decomposition is possible. A plan is described as a tree. A plan is not an ordered set of tasks since it is possible that some tasks are performed in parallel or several tasks are candidate but only one will be chosen. The plan model depicts plans as well as tasks as shown on Figure 2. Each plan contains the conditions under which this plan can be fired and the roles involved. We also add the plan substitutes in case this one cannot be fired.

The parallelism is rendered as an arc linking all the tasks as shown on the left bottom of Figure 2. A decision is rendered as a broken line linking all the tasks as shown on the right bottom of Figure 2. Tasks are ordered from left to right, from top to bottom. Agents begin by the first tasks at left. If this task has no further tasks linked to it, agents continue to the next task and so on.

A task is distinguished by its *name*. A natural language description is given in the compartment *goal*. The compartment *specification* gives a graphical description or an algorithm of this task. Graphical description can be found in PASSI [4]. The *preconditions* and *postconditions* define the conditions to be satisfied before and after the execution of the task. The *acceptance criteria* presents the set of acceptable solutions for the task. The last piece of information *task substitution* refers to the ability for agents to schedule another task if this task is not possible or will not be done. It is for instance the case when agents responsible for this task are on handheld devices and these ones are not accessible.

3.3. Interaction Model

Interaction is a key component in multiagent systems which allows agents to exchange information, cooperate and coordinate in order to complete their tasks. One usual method for representing interaction is a protocol. An agent interaction protocol is a set of rules that guide the interaction among several agents. For a given state of the protocol only a finite set of messages may be sent or received. If one agent is to use a given protocol, it must agree to conform to such a protocol and obey the various rules. Moreover, it must comply with its semantics. A thorough definition of interaction protocols can be found in [18].

The interaction model deals with the interaction between agents through protocols. The interaction model is decomposed in two parts: a textual part and a graphical part. The textual part corresponds to a requirement document for the protocol. The graphical part depicts the sequence of messages between agents or roles.

Our proposal of document is composed of several fields: protocol name, keywords, agents' roles, initiator, prerequisite, function, behavior, constraints, termination, security and mobility.

The field **Protocol's name** gives the name of the proto-

col. The field **Keywords** gives a list of keywords to characterize the protocol. The choice of keywords is free but it seems more interesting to provide accurate and meaningful keywords. Keywords could refer to

- the number of agents involved in the protocols,
- the kind of message sending: broadcast, multicast,
- the class of protocols: information request, negotiation, synchronization,
- the agent communication language used: ACL, KQML,
- the ontology used,
- the specific features: security, anonymity, fault-tolerance, electronic commerce

This list of keywords' domain is not exhaustive and depends on the purpose of the protocol.

The field **Agents' roles** refers to the agents involved in the protocol. It also gives the number of agents per role.

The field **Initiator** is related to the previous one. The initiator of an interaction is an agent playing a specific role. For instance, for auction protocols, the initiator is the seller. The initiator must be a role defined in the field *Agents' roles*.

The field **Prerequisite** defines the conditions to be satisfied before executing protocols. These conditions can be written as a free-format text or as a formal description. A possible formal description language is OCL [38] also employed for UML and especially for Agent UML sequence diagrams. The domain for the prerequisite usually depends of the interaction protocol context. For instance, if the interaction protocol deals with information request, the agent requesting must have been first identified by the agent having the information.

The field **Function** is used during reuse. This field gives a summary of the definition of the protocol. It seems to be interesting for designers in a context of reuse not to have to read the complete definition of the protocol to understand its purpose. The complete definition of the protocol is given in the field *Behavior*.

The field **Behavior** is the main field in the analysis document. It gives the complete definition of the protocol. Unlike communication protocol engineering where the execution paths are separated from message and data types, we put together these two pieces of information. We think it is better for designers to be able to refer to message type when they consider a path of the interaction. The requirement analysis document should also contain information which is related to agents and multiagent systems such as beliefs, desires and intentions (BDI) [41], actions or knowledge. Actually, sending or receiving messages can modify agents' behaviors. For instance, if we refer to FIPA ACL specifications [15], it is written that if an agent receives an *inform* message, it means that the sender believes the proposition and, the sender intends the receiver to believe this proposition. Moreover, when agents receive a *cfp* message, they have to verify if they are able to do the task. Thus, they have to do some actions.

The field **Constraints** allows designers to describe the good properties to be present in protocols and the bad properties to avoid. Good properties and bad properties refer to properties that designers want to check during the validation stage [20]. It might be deadlock freeness, termination, absence of acceptance cycles, absence of non-progressing states, mutual exclusion, etc. Constraints can be defined as a temporal logic formula. Thus, it is easier to check the properties with model checker such as SPIN [21]. This field can also contain the standards that protocols have to conform to. For instance, standards could be for instance the use of ACL for communicative acts or XML for message format.

The field **Termination** defines the valid termination of protocols. This field is related to the purpose of protocols. For instance, for a secure auction protocol, the seller has got the money but no longer the good; and the buyer has got the good but no longer the money. This field can be described through a free-format text or temporal logic formulae. Then, these formulae can be used in model checker for the validation.

The field **Security** gives the security requirements for this protocol. It could be that messages are encrypted. Other security requirements could be that agents are authenticated, or interactions are anonymous.

The field **Mobility** refers to mobility requirements. Such examples of mobility requirements are fault-tolerant interactions or message acknowledgement. Fault-tolerant interaction requirement deals with a more reliable media for message exchanges. It is particularly interesting when agents are on handheld devices or mobile appliances. Message acknowledgements correspond to the hand-shaking protocol in distributed systems. As long as agents do not acknowledge message receipt, the message is sent to the receivers.

The graphical part of the interaction model gives the sequence of messages between agents or roles. This graphical part presents some redundancy with the behavior field in the requirement analysis document but it is sometimes easier to read a diagram than to read a natural language description. We use the Agent UML sequence diagrams to this purpose [37] [1] as shown on Figure 3. We add couple of new elements for these sequence diagrams in [23].

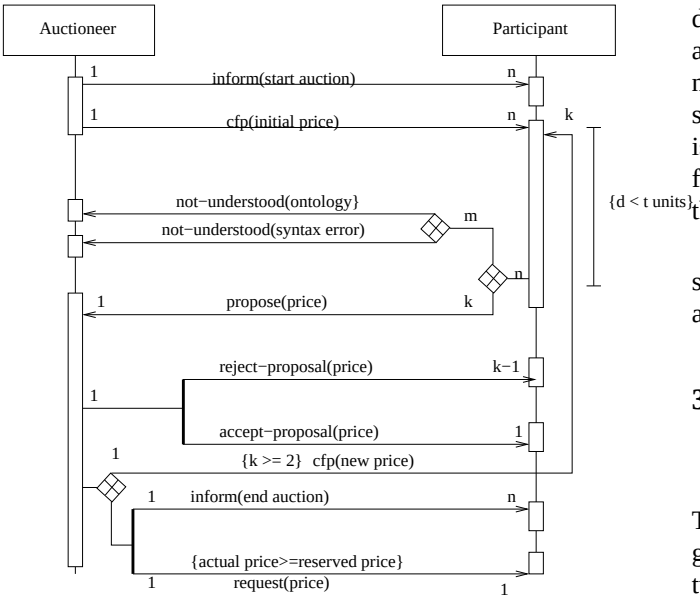

Figure 3. Agent UML Sequence Diagram for English Auction Protocol

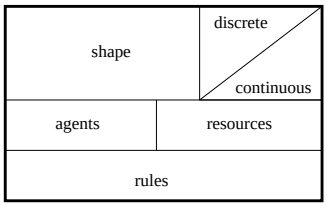

Figure 4. Environment Model Schema

3.4. Environment Model

Most of time, agents are included in a real or a virtual environment: the Internet, a computer, etc. The environment model describes the environment. The environment model needs to represent agents and resources. Such an example of environment model is in ROADMAP [27]. It does not cover the three dimensions of an environment but only the different elements in it.

The environment model is described as a schema as shown on Figure 4.

The environment model is composed of a *shape*. The shape is a free-format text describing the structure of the environment or a formula if the environment can be defined as a mathematical formula. An environment is *discrete* if there are a finite number of positions in it else the environment is *continuous*. The environment model describes the agents and the resources present in the environment. Agents are defined by their coordinates in the environment. Resources are also defined by their coordinates and the type and the number of resources are given. The *rules* deals with the consumption and the production of resources. This information is particularly important when the multiagent system is used for simulation like the Fishmarket project [34]. In this case, the resources are fish which appear and disappear.

The benefit of an environment model is when multiagent systems are applied to simulation and if agents are reactive agents.

3.5. Role Model

Agents within multiagent systems play specific roles. These roles correspond to their behaviors in this multiagent system. For instance, in auctions, there are usually two roles: auctioneer and participant. The role of auctioneer tries to sell items at the highest price possible. The role of participant bids for items and tries to buy items at the lowest price possible.

When security is at stake, it is important to define permissions and authorizations to agents according to their roles. Moreover, two roles can have two different levels of permissions. One role can access some specific pieces of knowledge and not the others. The notion of groups of roles is defined in Nemo and is based on Role-Based Access Control (RBAC) [13]. An application of RBAC to UML is in SecureUML [29]. The idea is the following one. Agents described through their roles are gathered into groups. Permissions and authorizations are associated to groups. These permissions and authorizations concern knowledge, actions or resources. For instance, if the piece of knowledge *salary* is only accessible to the group *management*, an agent must belong to this group to access it. The permissions on knowledge are *read*, *write* and *confidential*. The latter is more constraining than the first one, agents can read the piece of knowledge but are not allowed to pass it to other agents. Sub-groups into groups are possible. It is usually used to restrict access more or to define particular policies for a subgroup.

Group of roles, roles, the relationships between roles and permissions are given in the role model as shown on Figure 5.

A role is distinguished by its name. The role model also presents a natural-language *description* of the role, the *capabilities* and *knowledge*.

The *capabilities* compartment deals with the tasks the role is able to perform. This information is important in a context of open multiagent systems. New agents could be judged on the capabilities of their role. *Knowledge* depicts knowledge as well as the beliefs associated to this role.

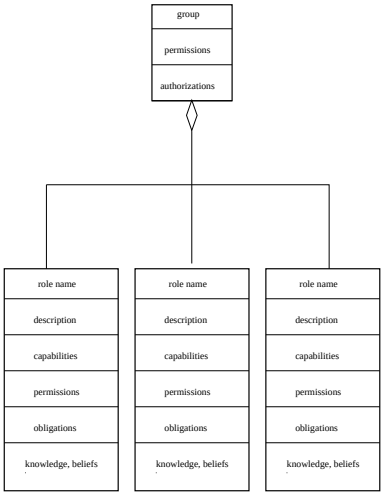

Figure 5. Role Model Schema

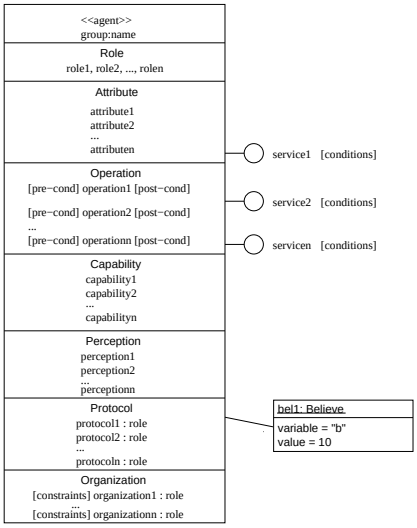

Figure 6. Agent Model Schema

3.6. Agent Model

Agents are the active entities in multiagent systems. Agents are complex to design since they contain perception, interaction, decision and knowledge. The Agent model proposed corresponds to the Agent UML class diagram proposal in [22].

The agent model is shown on Figure 6.

In comparison with the proposal in [22], we add some elements about mobility and security. Mobile agents have a different stereotype `<<mobile agent>>` instead of `<<agent>>`. Two elements for security are provided. First, the group of this agent is written before the name of the agent. Agent groups define the level of permissions and authorizations on knowledge, actions and resources (see Section 3.5). The second element of security is data privacy. It is possible to restrict the data access to a specific group of agents.

3.7. Knowledge Model

Knowledge is one of the main difference between agents and objects. Rational agents have beliefs, desires and intentions. Cognitive agents and rational agents have a model of their environment and of other agents. The knowledge model is similar to the one proposed in [7] except that we add the notion of permissions. It is then possible to restrict the access to a piece of knowledge to a group of agents. Moreover, knowledge can be restricted to *read access, write access and confidentiality access*. In the last case, agents can read but cannot reveal this piece of knowledge to other agents. Knowledge is rendered as a set of objects connected by relationships. Objects refer to UML objects as they are defined in object diagrams [2]. An object in UML is an instance of a class with well defined boundary. This approach allows designers to represent concrete data within agents. For instance, it is difficult to represent a belief such as *believe(Melbourne sunny)* directly within agent class particularly if we need to modify it after. As a consequence, it is easier to handle this information. When considering a goal, it is then possible to retrieve a part of the goal or to modify it. As far as we are concerned this approach brings flexibility to the management of BDI information. Moreover, it helps designers in the context of mutual beliefs and joint intentions [44], since it is easier to share beliefs or intentions if they are defined outside agents.

3.8. Deployment Model

The deployment model describes how entities involved in the multiagent systems are deployed on the network. This network can be a personal computer, a local area network or a wide area network such as the Internet. The deployment model is based on the UML deployment diagram [2]. Nevertheless, we are obliged to extend these deployment diagrams since they do not take into account the agent mobility. Some new stereotypes are added: `<<move>>` which means that an agent (or role) can move from one system to another one, `<<clone>>` which means that agents clone before leaving the current site and `<<change>>` which means that agents change roles when arriving at destination. They are placed on the dependencies. When the stereotype `<<change>>` is used, the new role is given after.

4. Lack of Related Work

As stated in introduction, several agent-oriented software engineering methodologies are on the shelf but as far as we know, no methodologies are considering mobility, security and open large scale multiagent systems together. The only work where mobility is considered is in [42] but it fails on the security management and in our opinion, mobility for multiagent systems cannot be considered without security.

Our main disadvantage in comparison with some other methodologies is that implementation is not considered. It is for instance the case in the PASSI methodology [4] and in the MaSE methodology [8]. Moreover, no tools are provided for the Nemo methodology.

Nevertheless, the Nemo methodology presents several advantages in comparison with other methodologies. For instance, the PASSI methodology [4] considers neither an environment model nor a knowledge model. Moreover, the agent definition is too close to UML class diagrams. As a consequence, it does not represent the richness and the complexity of agents.

The Tropos methodology [33] presents few models and is oriented to goal representation. Thus, it is not possible to represent organization, knowledge or environment.

Our final comparison is with the Gaia methodology [46]. In our opinion, this methodology is too abstract and does not present the richness of agents and multiagent systems. There is no model for organizations, for plans or for knowledge. The main advantage of Gaia is to address more accurately the notion of BDI systems and the formalization.

5. Conclusion and Future Work

Agent-oriented software engineering methodologies are a buzzword in agent theory and design. Several new methodologies emerged recently to tackle agent notions not provided by object-oriented methodologies. The new methodology Nemo has been presented in this paper. The particular aim of this methodology and its force is to address the problem of mobility, of security and of open large scale multiagent systems. These notions are barely present in current methodologies. The Nemo methodology focuses on the analysis and design stages. It is composed of eight models. The main model is the organization one. It presents some new ideas to tackle mobility, security and large scale multiagent systems.

The Nemo methodology is an ongoing research program as a consequence, several models have to be refined. Particularly, security has to be intertwined more deeply within models. We have already noticed several directions of work. The first one is to improve the reuse of models through projects. Nemo needs to allow designers to define templates.

The Nemo methodology contains eight models. It is a large number of models so it is important that designers have some help during design. We mean that when designers modify a model, they have to be informed what is the effect on other models. For instance, if designers reduce the responsibilities of a specific role due to security reason, it affects the role model as well as the plan model.

Obviously, designers need tools to help them. Finally, a particular effort has to be done on concrete examples. Concrete examples allow us to find flaws in the models and to advertise the methodology.

Acknowledgements. This research was supported by the UK government under EPSRC project GR/R27518 (Verifiable Languages and Protocols for Multiagent Systems).

References

[1] B. Bauer, J. P. Müller, and J. Odell. An extension of UML by protocols for multiagent interaction. In *International Conference on MultiAgent Systems (ICMAS'00)*, pages 207–214, Boston, Massachussetts, july, 10-12 2000.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unifi ed Modeling Language User Guide*. Addison-Wesley, Reading, Massachusetts, USA, 1999.

[3] F. Brazier, B. Dunin-Keplicz, N. R. Jennings, and J. Treur. Formal specifi cation of multi-agent systems: a real-world case. In V. Lesser, editor, *Proceedings of the First International Conference on Multi–Agent Systems*, pages 25–32, San Francisco, CA, 1995. MIT Press.

[4] P. Burrafato and M. Cossentino. Designing a multi-agent solution for a bookstore with the PASSI methodology. In *Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*, Toronto, Canada, May 2002.

[5] C. Caire, F. Garijo, J. Gomez, J. Pavon, F. Leal, P. Chainho, P. Kearney, J. Stark, R. Evans, and P. Massonet. Agent oriented analysis using MESSAGE/UML. In *Proceedings of Agent-Oriented Software Engineering (AOSE 01)*, Montreal, Canada, May 2001.

[6] A. Collinot and A. Drogoul. Using the Cassiopeia method to design a soccer robot team. *Applied Articial Intelligence (AAI) Journal*, 12(2–3):127–147, 1998.

[7] S. Cranefi eld. Networked knowledge representation and exchange using UML and RDF. *Journal of Digital Information*, 1(8), 2001.

[8] S. A. DeLoach. Multiagent systems engineering: a methodology and language for designing agent systems. In *Proceedings of Agent Oriented Information Systems '99 (AOIS'99)*, pages 45–57, Seattle, USA, May 1999.

[9] Y. Demazeau. *VOYELLES*. Habilitation diriger les recherches, Institut National Polytechnique de Grenoble, Grenoble, avril 2001.

[10] M. Esteva, J. A. Rodriguez, C. Sierra, P. Garcia, and J. L. Arcos. *On the formal specifi cations of electronic institutions*,

pages 126–147. Number 1991 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 2001.

[11] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, 2001.

[12] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98)*, pages 128–135. IEEE Computer Society, 1998.

[13] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[14] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Third International Conference on Information and Knowledge Management (CIKM-94)*. ACM Press, 1994.

[15] FIPA. *Specification*. Foundation for Intelligent Physical Agents, http://www.fipa.org/repository/fipa2000.html, 2000.

[16] M. S. Fox. An organizational view of distributed systems. *IEEE Trans. on System, Man, and Cybernetics*, 11(1):70–80, January 1981.

[17] N. Glaser. *Conceptual Modelling of Multi-Agent Systems The CoMoMAS Engineering Environment*, volume 4. Kluwer Academic Press, 2002.

[18] M. Greaves, H. Holmback, and J. Bradshaw. What is a conversation policy? In *Autonomous Agents'99 Special Workshop on Conversation Policies*, 1999.

[19] J. Habermas. *The Theory of Communicative Action*, volume 1 Reason and the Rationalization of Society. Beacon Press, Boston, 1984. transl. Mc Carthy Theorie des Kommunikativen Handels.

[20] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.

[21] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5), May 1997.

[22] M.-P. Huget. Agent UML class diagrams revisited. In B. Bauer, K. Fischer, J. Muller, and B. Rumpe, editors, *Proceedings of Agent Technology and Software Engineering (AgeS)*, Erfurt, Germany, October 2002.

[23] M.-P. Huget. Extending Agent UML protocol diagrams. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *AAMAS Workshop on Agent-Oriented Software Engineering (AOSE)*, Bologna, Italy, July 2002.

[24] C. Iglesias, M. Garrijo, J. Gonzales, and J. Velasco. Design of multi-agent system using MAS-CommonKADS. In Springer-Verlag, editor, *Proceedings of ATAL 98, Workshop on Agent Theories, Architectures, and Languages*, volume LNAI 1555, pages 163–176, Paris, France, July 1998.

[25] C. A. Iglesias, M. Garijo, and J. C. Gonzalez. A survey of agent-oriented methodologies. 1999.

[26] T. Juan, A. Pearce, and L. Sterling. Extending the Gaia methodology for complex open systems. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 02)*, Bologna, Italy, July 2002. ACM Press.

[27] T. Juan, L. Sterling, and M. Winikoff. Assembling agent oriented software engineering methodologies from features. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *Proceedings of Third International Workshop on Agent-Oriented Software Engineering (AOSE-2002)*, Bologna, Italy, July 2002.

[28] J. Lind. *Iterative Software Engineering for Multiagent Systems - The MASSIVE Method*. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2001.

[29] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In *Proceedings of the Fifth International Conference on Unified Modeling Language (UML 2002)*, Dresden, Germany, October 2002.

[30] A. Mali. Social laws for agent modeling. In M. Tambe and P. Gmytrasiewicz, editors, *Agent Modeling Papers from the AAAI Workshop*, pages 53–60. AAAI Press, 1996.

[31] J. Martin and J. Odell. *Object Oriented Analysis and Design*. Prentice-Hall, 1992.

[32] J. Müller, B. Bauer, and M. Berger. Software agents for electronic business: Opportunities and challenges. In V. M. et al., editor, *Proceedings of MASA 2001*, number 2322 in LNAI, pages 61–106. Springer, 2001.

[33] J. Mylopoulos, M. Kolp, and J. Castro. UML for agent-oriented software development: the tropos proposal. In *Proceedings of the Fourth International Conference on the Unified Modeling Language (UML 2001)*, Toronto, Canada, October 2001.

[34] P. Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autnoma de Barcelona, 1998.

[35] J. Odell. Agents and complex systems. *Journal of Object Technology*, 1(2), July-August 2002.

[36] J. Odell. Objects and agents compared. *Journal of Object Computing*, 1(1), May 2002.

[37] J. Odell, H. V. D. Parunak, and B. Bauer. Representing agent interaction protocols in UML. In P. Ciancarini and M. Wooldridge, editors, *Proceedings of First International Workshop on Agent-Oriented Software Engineering*, Limerick, Ireland, june, 10 2000. Springer-Verlag.

[38] OMG. UML 1.4. Technical report, OMG, 2001.

[39] L. Padgham and M. Winikoff. Prometheus: a methodology for developing intelligent agents. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *AAMAS Workshop on Agent-Oriented Software Engineering (AOSE)*, Bologna, Italy, July 2002.

[40] H. V. D. Parunak and J. Odell. Representing social structures in UML. In M. Wooldridge, G. Weiss, and P. Ciancarini, editors, *Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001)*, LNCS, Montreal, Canada, May 2001. Springer-Verlag.

[41] A. Rao and M. Georgeff. Modeling rational agents within a BDI architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R 91)*, pages 473–484, San Mateo, 1991. Morgan Kaufmann Publishers.

[42] A. Shelf. Design and specification of dynamic, mobile and reconfigurable multiagent systems. Master's thesis, Graduate School of Engineering and Management of the Air Force Institute of Technology, March 2001. AFIT/GCS/ENG/01M-11.

[43] G. Wagner. The agent-object-relationship metamodel: Towards a unified conceptual view of state and behavior. *Information Systems*, 2002. to appear.

[44] M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.

[45] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley and Sons, April 2002.

[46] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.