RIO : Roles, Interactions and Organizations

P. Mathieu, J.C. Routier, and Y. Secq

Laboratoire d'Informatique Fondamentale de Lille – CNRS UPRESA 8022
UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
59657 Villeneuve d'Ascq Cedex
{mathieu,routier,secq}@lifl.fr

Abstract. The notions of role and organization have often been emphasized in several agent oriented methodologies. Sadly, the notion of interaction has seldom been reified in these methodologies. We define here a model of runnable specification of interaction protocols. Then, we propose a methodology for the design of open multi-agent systems based on an engineering of interaction protocols. These interaction protocols are described in term of conversation between micro-roles characterized by their skills, then micro-roles are gathered in composite roles. Then, composite roles are used to build abstract agents. Lastly, these latter can be distributed on running agents of a multi-agent system.

1 Introduction

The idea of an agent based software engineering has appeared roughly ten years ago, with the paper from Shoham entitled *Agent Oriented Programming*[13]. Since these days, several methodologies have been proposed to help developers in their analysis and design[6, 2]. For that, the concepts of role, interaction and organization are often proposed to facilitate the decomposition and the description of distributed systems. However, we think that suggested methodologies do not clearly identify the various levels of abstraction making it possible to break up a system and especially they generally do not propose pragmatic concepts or principles facilitating the realization of such systems. Thus, our proposal relies on a model of minimal generic agent and a model of executable specification of interactions. The agent model is the infrastructure allowing the deployment and the management of interactions, while the specification of the interactions describes a global sight of the conversations between the agents of the system.
In the first part of this article, we briefly present two methodologies which were proposed for the use of multi-agent systems for the design of complex distributed systems, then we put them in relation with interaction oriented approaches. In the second part, we propose a model of minimal generic agent and a formalism for the specification of interaction protocols between micro-roles. The latter are assembled in composite roles which are then atributed to the agents of the system. This specification is made executable by the generation of Colored Petri

Nets for each micro-role. This executable specification and the use of a generic model of agent enable us to propose the RIO methodology facilitating the design, the realization and the effective deployment of multi-agent systems.

2 Agent methodologies and interaction languages

Several agent oriented methodologies have been proposed like Aalaadin[3] or Gaia [16]. It is significant to notice that these two methodologies do not make any assumption on agent models and concentrate mainly on the decomposition in term of roles of a complex system. This point is fundamental, in particular because of the multiplicity of available agent and multi-agent systems models. This multiplicity makes the task of the developer difficult: which agent model should be used? Which organizational model should be chosen? Indeed, each platform imposes too often both its own agent model and its organizational model. These methodologies are interesting on many points, but remains too general to ease the transition from the design stage to its concrete realization. Moreover, the various levels of communication are not clarified in the description of interaction protocols. Indeed, works on agent communication languages (ACL)identify three levels that constitutes a conversation: the semantic, the intention (these two are expressed through languages like KIF or SL, and KQML or FIPA-ACL), and the interaction level. However, even by considering heterogeneous platforms sharing the same ontology, it remains difficult to have guarantees on the respect of interaction protocols. This is the reason why works have been undertaken to formalize this aspect with several objectives: to describe the sequence of the messages, to have certain guarantees on the course of a conversation and to ease interoperability between heterogeneous platforms.

Interaction languages. To illustrate these approaches based on a formalization of the interactions, we studied three of them: April[9], AgenTalk[5] and COOL[1]. April is a symbolic language designed to handle concurrent processes, that eases the creation of interaction protocols. In April, the developer must design a set of *handlers* which treats each message matghing a given pattern. According to the same principles, AgenTalk adds the possibility to create easily new protocols by specialization of existing protocols, by relying on a subclassing mechanism. Another fundamental contribution of AgentTalk is the explicit description of the protocol: the conversation is represented by a set of states and a set of transition rules. This same principle was employed in COOL, which proposes to model a conversation using an finite state automata. In COOL, the need for the introduction of *conventions* between agents to support coordination is proposed. This concept of *convention* must be brought closer to the works of Shoham on *social rules* [14] and their contributions on the global performance of the system. Thus, the introduction of this level of interaction management while rigidifying in a certain way the possible interactions between agents, brings guarantees on coordination and allows the reification of these interactions. The table below, inspired by work of Singh[15], illustrates the various levels of abstractions within a multi-agent system:

Applicative skills	Business knowledge
Agent models and system skills	Agent oriented design
Conversation management	Interaction oriented design
Message transport	Agent platform (i.e. agents container)

To conclude, we would like to cite a definition suggested by Singh[15] of the interaction oriented approach, which characterizes our approach: *We introduce interaction-oriented programming (IOP) as an approach to orchestrate the interactions among agents. IOP is more tractable and practical than general agent programming, especially in settings such as open information environments, where the internal details of autonomously developed agents are not available.* It is the point of view that we adopt, by proposing a pragmatic method for the design and realization of multi-agent systems, relying on the concept of *executable* specification of interaction protocols.

3 Interaction oriented design

The heart of our proposal is a formal model to describe interaction protocols, and a transformation mechanism to generate the code that is necessary to the management of these protocols. In order for running agents to be able to exploit these new interactions, we rely on a minimal generic agent model, which authorizes the incremental construction of agent per skills addition. Thus, we will initially present this generic agent model, then we will study the model of specification of interaction protocols and the associated transformation mechanism.

4	Applicative skills	Database access, graphical user interface ...
3	Agent model related skills	inference engine, behavioral engine, ...
2	Agenthood skills	Knowledge base, conversation management, organizations management
1	Minimal system skills	Communication and skill management

Table 1. The four layer of our abstract agent model

A minimal generic agent model. The basis of our model is on the one hand the interactive creation of agent, and on the other hand a search on the fundamental functionalities of agenthood. We are not interested in the description of the individual behavior of agents, but rather in the identification of functions that are sufficient and necessary to an agent. Indeed, the management of interactions, the knowledge management or the management of organizations, are not related to the agent model, but are intrinsic characteristics with the concept of agent. In our model, an agent is a container which can host skills. A skill is a coherent set of functionalities accessible through a neutral interface. This concept of skill is to be brought closer to the concept of software component in object oriented technologies. Thus, an agent consists of a set of skills which carries out

various parts of its behavior. We identified four layers which are characterized by the various levels of abstraction of functionalities that are proposed (table 1). The first level corresponds to "system" skills, i.e. the minimal functionalities allowing to bootstrap an agent: the communication (emission/reception of messages) and the management of skills (dynamic acquisition/withdrawal of skills)[12]. The second level identifies *agent* skills: the knowledge base, media of interaction between skills and the place of knowledge representation, the management of interaction protocols (cf. following section) and the management of organizations (cf. last section). The third level is related to skills that define the agent model (reactive, BDI...), while the last level represents purely applicatives skills. Thus, the first and the second level characterize our generic minimal agent model. This model is generic with respect to the agent models that can be used, and minimal in the sense that it is not possible to withdraw one of the functionalities without losing a fundamental aspect of agenthood.
A skill is made of two parts: its *interface* and its *implementation.* The interface specifies the incoming and outgoing messages, while the implementation carries out the processing of these messages. This separation uncouples the specification from its realization, and thus makes it possible to have several implementations for a given interface. The interface of a skill is defined by a set of message patterns which it accepts and produces. These messages must be discriminated, it is thus necessary to type them.

`interface := ((m`$_{in}$`)+, (m`$_{out}$`)*)* where m`$_x$` = message pattern`

The typing of message patterns can take several forms: a strong typing, which has the advantage of totally specifying the interfaces, while a weak typing offers more flexibility with regard to the interface evolution. Thus, if the content of messages are expressed in KIF or DAML+OIL, a strong typing will consist of an entire message checking, while a weak typing will only check it partially.

A model of executable specification of interaction protocol. Many works have been done to specify interaction protocols. Recently, AgentUML[10] was defined like an extension of UML, to specify the conversations between agents, in particular by specializing sequence diagrams in UML. However, these specifications require the interpretation of developers, which must then translate them in their own system. Works of Labrou and Finin[11] explore the use of Colored Petri Nets[4] (CPN) to model conversations between agents. In [8], the same approach is used, but the concept of Recursive Colored Petri Nets is introduced to support conversations composition. Our work follows the same principles: to represent interactions in a global way, and to use a recognized and established formalism. However, contrary to preceding works, our goal is to produce an executable *specification.* i.e., a description of the interaction protocol which can be then directly integrated in a running system. Moreover, CPN are unquestionably adapted to the modeling of concurrent processes, and provide an interesting graphic formalism, but they are unfortunately not really user friendly. This is why it appears preferable to us to define a language adapted to the modeling of interaction protocols, and to use a projection mechanism that translates this language into CPN.

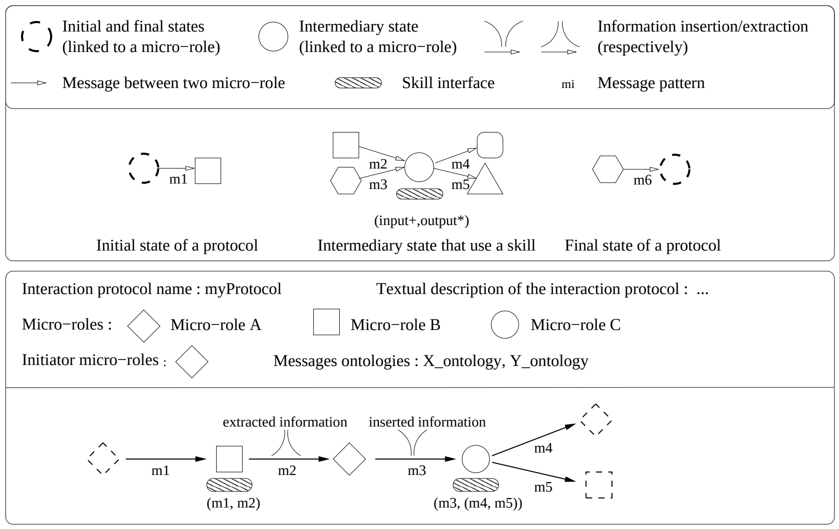

Fig. 1. Definition of the syntactic elements that constitute interaction protocols

The interaction protocol specification model. The purpose of this model is to ease the specification, the checking and the deployment of interaction protocols within multi-agent systems. On all these stages, the designer has to define the specification, the other stages being automated. For that, we define a formalism representing the global view of an interaction protocol, and a projection mechanism which transforms this global view into a set of local views dedicated to each role. We will initially describe the specification of the global view, before presenting the projection mechanism which generate local views that agents use while a protocol is running. The interaction protocols are regarded here as social laws within the meaning of Shoham[14], that means that agents lose part of their autonomy (conversational rules are static), but the system gains in determinism and in reliability. Our model relies on the concept of skill, micro-role and a graph that represents the state of the conversation. An interaction protocol formally specifies the course of a conversation (regarded as a social law) between various entities, i.e. the nature of exchanged messages, the flow of these messages and skills that entities must implement for each stage of the conversation. These entities correspond to micro-roles, and are characterized by their name and their skills. One uses a graph to represent the course of the conversation: the nodes represent micro-roles which can be associated to a skill interface, and the arcs with a sending of message between two micro-roles (typed by a message pattern). An interaction protocol is thus defined by the following elements (figure 1): the name of the interaction protocol, a textual description of the goal of this protocol, the list of the micro-roles involved in the interaction and the geometrical symbol which is associated to them, ontologies of exchanged messages, a list of information necessary in input and produced at output, an interaction graph gathering information like the temporal course of the conversation, the synchronization and the nature of the exchanged messages, and the skills that are used by micro-roles. What it is significant to understand is that the designer has a global view of the interaction: the flow of the messages, their nature (the type of these messages corresponds to the annotations attached to the arcs), needed skills and information used or produced. In addition, all information necessary to the management of the interaction protocol is centralized here and can be used to carry out the generation of the code required to manage this interaction for each micro-role. The designer has thus only to define the interaction protocol by using a graphical tool, the projection mechanism takes care of the generation of descriptions for each micro-role, and the generic agent model can then use these descriptions.

The projection mechanism. The preceding section described the formalism representing interaction protocols, we will now explain the transformation making it possible to obtain a *runnable* specification. The specification of interaction protocols gives to the designer a global view of the interaction. Our objective is to generate for each micro-role a local view starting from this global view, this one could then be distributed dynamically to the agents of the system. The projection mechanism transforms the specification into a set of automata. More precisely, an automata is created for each micro-role. This automata manages the course of the protocol: coherence of the protocol (messages scheduling), messages types, side effects (skill invocation). The implementation of this mechanism is carried out by the generation of Colored Petri Nets. Indeed, we use the color of tokens to represent messages patterns, in addition we have a library facilitating the interactions between generated networks and the agent skills. On the basis of the interaction graph, we create a description of Colored Petri Net for each micro-role, and we transform this textual description to a Java class. This class is then integrated within a skill, which is used by conversation manager skill (level 2 in table 1). The interest of this approach is that the designer graphically specifies the global view of the interaction, the projection mechanism generates the skill needed to the management of this interaction. Moreover, thanks to the dynamic skill acquisition, it is possible to add new interaction protocols to running agents of the system.

Knowledge and organization management. The knowledge management and the management of organizations are also mandatory functionnalities of agent, and they should not be enclosed within the agent model. The knowledge management gathers at the same time their representation, the information storage, the means of reaching and of handling them. The implementation of these functionalities is strongly dependent on the used agent model (third level of table 1). The concept of organization is necessary to structure interactions that intervene between entities of the system. This concept brings some significant benefits: a means to logically organize the agents, a communication network per defect and a media to locate agents, roles or skills. Moreover, its reification provides a door in the system, making it possible to visualize and to improve interactions between agents[7].

4 RIO : towards an interaction based methodology

In this section, we will present the methodology that we are developing, and which relies on the previously presented concept of *runnable* specification . This methodology falls under the line of GAIA[16], and thus aims the same applicability. However, GAIA remains too general to easily be able to go from the system design stage to its realization. The purpose of our proposal is to facilitate this transition. The RIO methodology relies on four stages, the two first represent reusable specifications, while the two last are singular with the application (figure 2). Moreover, we will not speak about *application*, but about an agent society. Indeed, the RIO methodology proposes an incremental and interactive construction of multi-agent systems. By analogy with our minimal generic agent model, where an agent is a container which can receive skills, we see a multi-agent system like a container that has to be enriched by interactions. We will detail this approach by studying the four stages of our methodology.

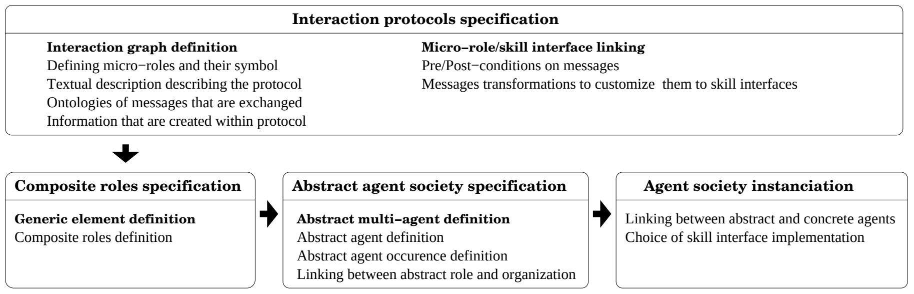
Interaction protocols specification
Interaction graph definition
Defining micro−roles and their symbol
Textual description describing the protocol
Ontologies of messages that are exchanged
Information that are created within protocol
Micro−role/skill interface linking
Pre/Post−conditions on messages
Messages transformations to customize them to skill interfaces
Composite roles specification
Generic element definition
Composite roles definition
Abstract agent society specification
Abstract multi−agent definition
Abstract agent definition
Abstract agent occurence definition
Linking between abstract role and organization
Agent society instanciation
Linking between abstract and concrete agents
Choice of skill interface implementation

Fig. 2. The stages of the RIO methodology

Interaction protocols specification. The first stage consists in identifying the involved interactions and roles. Then, it is necessary to determine the granularity of these interactions. Indeed, for reasons of re-use of existing protocols, it is significant to find a balance between protocols using too many roles, these protocols becoming thus too specific, and protocols where there are only two roles, in this case the view of the interaction is no more global. The specification of the interaction protocols can then be done in three ways : either *ex-nihilo*, by specialization, or by composition. Creation *ex-nihilo* consists in specifying the interaction protocol by detailing its cartouche (figure 1). Specialization makes it possible to annotate an existing cartouche. Thus, it is possible to specify the cartouche of an interaction protocol such as FIPA ContractNet, its specialization will consist in changing micro-roles names to adapt them to the application, to refine message patterns, and if required to modify insertions/extractions of information. Finally, the composition consists in assembling existing protocols by specifying associations of micro-roles and information transfers. At the end of this stage, the designer has a set of interaction protocols. He can then pass to the description of composite roles, which will allow the aggregation of micro-roles that are involved in complementary interactions.

Composite roles specification. This second stage specifies role models. These models are abstract reusable descriptions. The composite roles correspond to a logical gathering of micro-roles. These patterns define *abstract* roles, which gather a set of consistent interaction protocols. For example, a composite role SUPPLIES MANAGEMENT will gather the micro-role BUYER within the PROVIDERS SEEKING interaction protocol and the micro-role STOREKEEPER of the interaction SUPPLIES DELIVERY. Indeed, a role is generally composed of a set of tasks which can be, or which must be carried out by the agent playing this role. Each one of these tasks can be broken up and be designed as a set of interactions with other roles. The concept of composite role is thus used to give a logical coherence between the micro-role representing the many facets of a role.

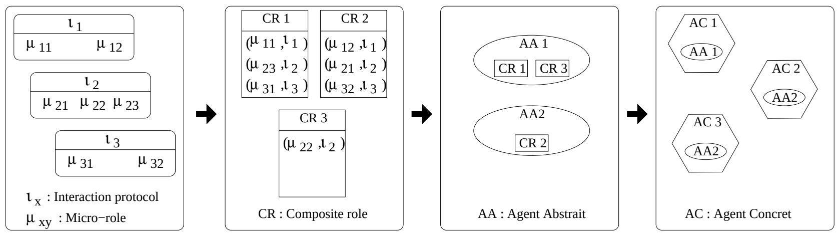

Fig. 3. Synthetic illustration of RIO stages

Agent societies specification. This third stage can be regarded as a specification of an abstract agent society, i.e. a description of abstract agents and their occurrence, as well as the link between composite roles and organizations. Once the set of composite roles is created, it is possible to define the abstract agents (patterns of agent, or *agent template*), which are defined by a set of composite roles. These abstract agents describe applicative agent models. These models are specific, because they introduce strong dependencies between composite roles. For example, the OFFICE STATIONERY DELIVERY composite role could be associated with the TRAVELLING EXPENSES MANAGEMENT composite role to characterize a LABORATORY SECRETARY abstract agent (fig 4). Once abstract agents are defined, it is necessary to specify their occurrence in the system. It means that each abstract agent has an associated cardinality constraint that specifies the number of *instances* that could be created in the system (exactly N agents, 1 or more, *, or [m..n]). The second part of this stage consists in specifying for each abstract agent, and even for the composite roles of these agents, which organization should be used to find their acquaintances. Indeed, when agents are running, they have to initiate interaction protocols, but in order to do that they initially have to find their interlocutors. The organization is used as a media for this search. This association makes it possible to use various organizations for each interaction protocol.

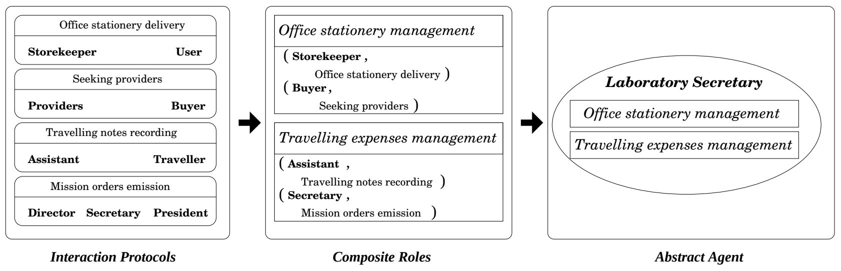
Office stationery delivery
Storekeeper
User
Seeking providers
Providers
Buyer
Travelling notes recording
Assistant
Traveller
Mission orders emission
Director
Secretary
President
Interaction Protocols
Office stationery management
(Storekeeper , Office stationery delivery)
(Buyer , Seeking providers)
Travelling expenses management
(Assistant , Travelling notes recording)
(Secretary , Mission orders emission)
Composite Roles
Laboratory Secretary
Office stationery management
Travelling expenses management
Abstract Agent

Fig. 4. The Secretary example

Instantiating an agent society in a multi-agent system. This last stage specifies deployment rules of the abstract roles on the running agents of a system. We have a complete specification of the agent society, that can be mapped on the concrete agents of the multi-agent system. For that, it is necessary to indicate the assignments from abstract agents to concrete ones. Then, the connection between a skill interface and its implementation is carried out. The designer indeed must, according to criteria that are specific to the hosting platform or applicative, bind the implementation with skill interfaces. It is during deployment that the generic agent model is justified as a support to dynamic acquisition of new skills related with the interaction. Indeed, the interaction, once transformed by the projection mechanism, is represented for each micro-role by a Colored Petri Net and its associated skills. All these information are sent to the agent, which adds applicative skills and delegates the CPN to the conversation manager. When an agent receives a message, the conversation manager checks if this message is part of a conversation in progress (thanks to the conversation identifier included in the message), if it is the case, it delegates the message processing to the concerned CPN, if not he seeks the message pattern matching the received message and instantiates the associated CPN. If it does not find any, the message will have to be treated by the agent model.

5 Conclusion

We have presented in this article a methodology falling under the line of GAIA, but relying on the concepts of the interaction oriented programming. The basis of the RIO methodology is the engineering of interaction protocols, and more precisely the engineering of *runnable* specifications. For that purpose, we use a tool facilitating the graphical design of interaction protocols, and a projection mechanism that generates the code corresponding to the vision that each participant has of the interaction. By using these specifications, it is possible to create abstractions characterizing the various roles and agents of a multi-agent system: the composite roles, which gather a set of micro-roles, and abstract agents, which gather a set of composite roles. An implementation of this approach is under development, and we use the following technologies: Coloured Petri Nets, DAML+OIL for messages ontologies and knowledge representation, OSGi as component model, and the Java language for the multi-agent platform.

References

1. M. Barbuceanu and M. S. Fox. Cool: A language for describing coordination in multiagent systems. In *Proceedings of the First International Conference oil Multi-Agent Systems (ICMAS-95)*, pages 17–24, San Francisco, CA, 1995.
2. F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, and J. Treur. DESIRE: Modelling multi-agent systems in a compositional formal framework. *Int Journal of Cooperative Information Systems*, 6(1):67–94, 1997.
3. J. Ferber and O. Gutknecht. Operational semantics of a role-based agent architecture. In *Proceedings of ATAL'99*, jan 1999.
4. Kurt Jensen. Coloured petri nets - basic concepts, analysis methods and practical use, vol. 1: Basic concepts. In *EATCS Monographs on Theoretical Computer Science*, pages 1–234. Springer-Verlag: Berlin, Germany, 1992.
5. Nobuyashu Osato Kazuhiro Kuwabara, Toru Ishida. Agentalk : Describing multi-agent coordination protocols with inheritance.
6. E. A. Kendall, M. T. Malkoun, and C. H. Jiang. A methodology for developing agent based systems. In Chengqi Zhang and Dickson Lukose, editors, *First Australian Workshop on Distributed Artificial Intelligence*, Canberra, Australia, 1995.
7. P. Mathieu, J.C. Routier, and Y. Secq. Principles for dynamic multi-agent organisations. In *Proceedings of Fifth Pacific Rim International Workshop on Multi-Agents (PRIMA2002)*, August 2002.
8. Hamza Mazouzi, Amal El Fallah Seghrouchni, and Serge Haddad. Open protocol design for complex interactions in multi-agent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 517–526. ACM Press, 2002.
9. Frank G. McCabe and Keith L. Clark. April – agent process interaction language. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI volume 890)*, pages 324–340. Springer-Verlag: Heidelberg, Germany, 1995.
10. J. Odell, H. Parunak, and B. Bauer. Extending uml for agents, 2000.
11. Tim Finin Yannis Labrou R. Scott Cost, Ye Chen and Yun Peng. Modeling agent conversations with colored petri nets. In *Third Conference on Autonomous Agents (Agents-99), Workshop on Agent Conversation Policies*, Seattle, May 1999. ACM Press.
12. JC. Routier, P. Mathieu, and Y. Secq. Dynamic skill learning: A support to agent evolution. In *Proceedings of the AISB'01 Symposium on Adaptive Agents and Multi-Agent Systems*, pages 25–32, 2001.
13. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.
14. Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73(1-2):231–252, 1995.
15. Munindar P. Singh. Toward interaction-oriented programming. Technical Report TR-96-15, 16, 1996.
16. M. Wooldridge, NR. Jennings, and D. Kinny. The GAIA methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 2000.