A Verification by Abstraction Framework for organizational Multi-Agent Systems

Nicolas Gaud, Vincent Hilaire, Stéphane Galland, Abderrafiâa Koukam, and Massimo Cossentino

Multiagent Systems Group,
System and Transport Laboratory
University of Technology of Belfort Montbéliard
90010 Belfort cedex, France
{nicolas.gaud,vincent.hilaire,
stephane.galland,abder.koukam}@utbm.fr
http://set.utbm.fr

Abstract. Software agents and multi-agents systems (MAS from now on) are recognized as both abstractions and effective technologies for modelling and building complex distributed applications. However, they are still difficult to engineer. The reason is that when massive number of autonomous components interact it is very difficult to predict that the emergent organizational structure fits the system goals or that the desired functionalities will be fulfilled. Verification approaches try to evaluate whether or not a product, service, or system complies with a specification. However verification approaches are limited by the state-space of the system under study. This paper proposes an approach based upon an organizational framework and specifically the capacity concept which enables to abstract a role know-how and to reduce the state space of the system under study. A formal framework based on multi-formalisms language and the specification approach are presented and illustrated through the specification of a part of the contract net protocol.

Key words: Agent Oriented Software Engineering, Verification, Abstraction

1 Introduction

Software agents and multi-agents systems (MAS from now on) are recognized as both abstractions and effective technologies for modelling and building complex distributed applications. However, they are still difficult to engineer. The reason is that when massive number of autonomous components interact it is very difficult to predict that the emergent organizational structure fits the system goals or that the desired functionalities will be fulfilled. Verification approaches try to to evaluate whether or not a product, service, or system complies with a specification. However verification approaches are limited by the statespace of the system under study. In order to tackle this problem and to verify properties

for large systems such as MAS there are several techniques. One of these techniques is verification by abstraction. It consists in finding an abstraction relation and an abstract system that simulates the concrete one and that is amenable to algorithmic verification [3, 16].
The goal of this paper is to present a verification by abstraction approach dedicated to MAS and specifically organizational MAS and Holonic MAS (HMAS). This approach is based upon the abstraction of capacities of roles played by agents within organizations. Organizational approaches are now common within the MAS domain [11, 20, 2, 4] and propose organizational concepts for MAS and HMAS modelling. The framework presented in this paper, namely CRIO, is based upon four main concepts : Capacity, Role, Interaction and Organization. Agents play roles within organizations and interact between themselves. In order to be played by an agent, a role may require some capacities. A capacity is an abstraction of a know-how or a service. It is a very useful concept during the analysis and design of HMAS [18]. The verification by abstraction approach presented here is based upon this concept. Each capacity abstracts a part of role behaviours and separate it from it current implementation.
Each concept of the CRIO framework is specified using a formal language namely OZS [13]. This language composes two formalisms, Object-Z [10] and statecharts [14]. The formal semantics defined for this notation allows the verification of properties by using dedicated software environment such as SAL [7].
This paper is organized as follows, section 2 introduces OZS notation. Section 3 presents the CRIO framework, section 4 illustrates the framework and the abstraction approach using the contract net protocol. Eventually, section 5 concludes.

2 Background

Many specification formalisms can be used to specify entire system but few, if any, are particularly suited to model all aspects of such systems. For large or complex systems, like MAS, the specification may use more than one formalism or extend existing formalism.
Our choice is to use Object-Z to specify the transformational aspects and statecharts to specify the reactive aspects. Object-Z extends Z [17] with object-oriented specification support. The basic construct is the class which encapsulates state schema and operation schemas which may affect its variables.
Statecharts extend finite state automata with constructs for specifying parallelism, nested states and broadcast communication for events. Both language have constructs which enable refinement of specification. Moreover, statecharts have an operational semantic which allows the execution of a specification.
We introduce a multi-formalisms notation that consists in integrating statecharts in Object-Z classes. The class describes the attributes and operations of the objects. This description is based upon set theory and first order predicates logic. The statechart describes the possible states of the object and events which may change these states. A statechart included in an Object-Z class can use attributes

and operations of the class. The sharing mechanism used is based on name identity. Moreover, we introduce basic types $[Event, Action, Attribute]$. *Event* is the set of events which trigger transitions in statecharts. *Action* is the set of statecharts actions and Object-Z classes operations. *Attribute* is the set of objects attributes.
The *LoadLock* class illustrates the integration of the two formalisms. It specifies a *LoadLock* composed of two doors which states evolve concurrently. Parallelism between the two doors is expressed by the dashed line between $DOOR1$ and $DOOR2$. The first door reacts to $activate1$ and $deactivate1$ events. When someone enter the *LoadLock* he first activate the first door enter the *LoadLock* and deactivate the first door. The transition triggered by $deactivate1$ event execute the *inLL* operation which sets the *someoneInLL* boolean to true. Someone which is between the first and the second door can activate the second door so as to open it.

LoadLock

$someoneInLL : \mathbb{B}$

INIT

$\neg\ someoneInLL$

inLL

$\Delta someoneInLL$

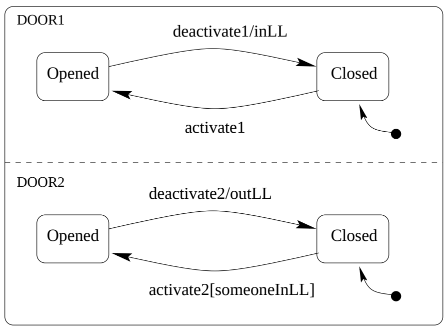
$someoneInLL'$

outLL

$\Delta someoneInLL$

$\neg\ someoneInLL'$

behavior



The notation for attribute modification consists of the modified attributes which belongs to the Δ-list. In any operation sub-schema, attributes before their modification are noted by their names and attributes after the operation are suffixed by '.

The result of the composition of Object-Z and statecharts seems particularly suited in order to specify MAS. Indeed, each formalism has constructs which enable complex structure specification. Moreover, aspects such as reactivity and concurrency can be easily dealt with.

3 CRIO

The CRIO metamodel presented in figure 1 is the basis of the framework we present in this paper. A more complete description of the metamodel related to a MAS methodology is given in [6]. The metamodel introduces two levels of abstraction.
The abstract level is concerned with the analysis of a problem in organizational terms. In order to describe the analysis the concepts chosen are role, interaction, organization and capacity. The adopted definition of role comes from [9]: "*Roles identify the activities and services necessary to achieve social objectives and enable to abstract from the specific individuals that will eventually perform them. From a society design perspective, roles provide the building blocks for agent systems that can perform the role, and from the agent design perspective, roles specify the expectations of the society with respect to the agent's activity in the society*". However, in order to obtain generic models of organizations, it is required to define a role without making any assumptions on the agent which will play this role. To deal with this issue the concept of capacity was defined [18]. A capacity is a pure description of a know-how. A role may require that individuals playing it have some specifics capacities to properly behave as defined. An individual must know a way of realizing all required capacities to play a role. Interactions are sequences of actions which consequences have influences over the future behaviours of roles. These interactions abstract interactions between agents and are described by the interaction concept of the CRIO metamodel. The context of these interactions is given by an organization. An organization is then a description of a set of roles and theirs interaction.
The concrete level describes the solution in terms of groups instantiating organizations. Entities belonging to groups, agents or holons, have capacity implementations required by the played roles. For each concept of this metamodel. A formal description using the OZS notation is given. These specifications define a framework that can be used to formally describe a MAS model. In this paper we will not give the specifications of group and holon which are not necessary for the example. The following types are defined and have to be refined : $[Attribute]$, $[Event]$ and $[Action]$ these types define respectively the sets of attributes, stimulus and actions of roles. The first concept specified is the role. The role class defines an empty behavior schema. It is to be refined to specify the behavior of the role. It will be specified by using a statechart. A role is also composed of a set of *attributes*, a set of *events* it can react to and a set of *actions*. The role is also defined by a set of *capacities* required by the role and the conditions that have to be met in order to play and leave the role. The constraint states that whatever stimulus (resp action) of the stimulus (resp actions) set it must

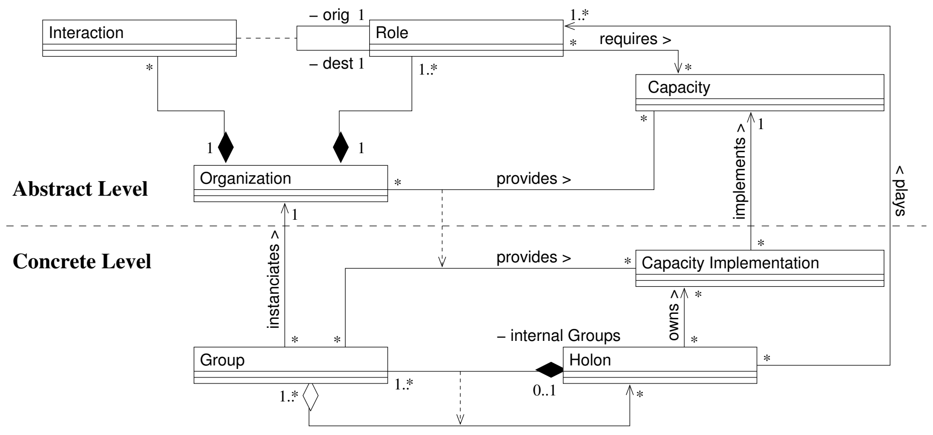


Fig. 1. CRIO meta-model

be present on at least one transition of the statechart defining the role behavior.

Role

behavior

$attributes : \mathbb{P}\, Attribute$
$stimulus : \mathbb{P}\, Event$
$actions : \mathbb{P}\, Action$
$requiredCapacities : \mathbb{F}\, Capacity$
$obtainConditions, leaveConditions : Condition$

$\forall\, s \in stimulus, \exists\, e \in behavior.\rho \bullet$
$\quad (\exists\, t \in e.transitions \bullet t.label.event = s)$
$\forall\, e \in behavior.\rho \bullet$
$\quad (\forall\, t \in e.transitions \bullet t.label.action \subseteq actions)$

An interaction is specified by a couple of roles, *orig* and *dest*, which are respectively the origin and the destination of the interaction. The roles *orig* and *dest* interact by the way of operations op_1 and op_2. These operations are combined by the $\|$ operator which equates output of op_1 and input of op_2. In order to extend interaction to take into account more than two roles or more complex interactions involving plan exchange one has to inherit from *Interaction*.

Interaction

$$\begin{array}{l} orig, dest : Role \\ op_1, op_2 : Action \\ \hline op_1 \in orig.action \\ op_2 \in dest.action \\ \hline \Diamond(orig.op_1 \| dest.op_2) \end{array}$$

Organization

$$\begin{array}{l} roles : \mathbb{P}\, Role \\ interactions : \mathbb{P}\, Interaction \\ \hline \forall\, i \in interactions \bullet \\ \quad (i.orig \in roles \land i.dest \in roles) \end{array}$$

An organization is specified by a set of roles and their interactions. Interactions happen between roles of the concerned organization. It is to say that for each interaction of the *interactions* set the roles of the interaction must belong to *roles* set of the organization. Moreover, each role must be part of at least one interaction.
The capacity class specifies the concept of capacity. This concept is described by a *name*, a set of attributes taken as *input* by the capacity and a set of *outputs* produced by the capacity. The *requires* and *ensures* sets of constraints specifies what must be true before the capacity can be called and after the capacity is called. This property is expressed with the constraint that whenever the capacity is called and the requires constraints are true then eventually the ensures constraint will be true.

Capacity

$$\begin{array}{l} name : String \\ inputs : \mathbb{F}\, Attribute \\ outputs : \mathbb{F}\, Attribute \\ requires : \mathbb{F}\, Constraint \\ ensures : \mathbb{F}\, Constraint \\ \hline capacityCall(name) \land (\bigwedge_{r \in requires}) \Rightarrow \Diamond(\bigwedge_{e \in ensures} e) \end{array}$$

A capacity implementation is specified by the CapacityImpl class. This class has an *implements* attributes that specifies which capacity it implements. The behaviour schema specifies how the capacity is implemented.

CapacityImpl

$$\begin{array}{l} implements : Capacity \end{array}$$

behaviour

With this framework one can specify a MAS or HMAS solution using organizational concepts. The next section describes a part of the contract net protocol specified using this framework.

4 Contract NET example

4.1 Specification

In this section the contract net protocol [19] is specified with the CRIO framework. We adopt the FIPA description of the contract net protocol [12]. The organization describing the contract net protocol is sketched in figure 2. This organization is composed of two roles : initiator and participant. The initiator is the manager who is interested in delegating a task. The participants are the members of the network which can receive the call for proposal and make propositions to the initiator.

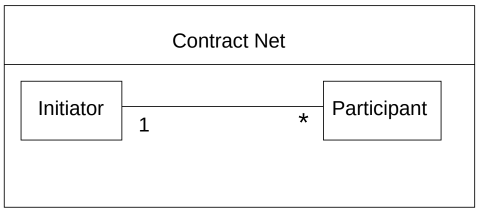


Fig. 2. Contract Net organization

The Initiator class specifies the Initiator role. It inherits from the role class of the CRIO framework and adds the following attributes : *proposals* which is a set of Proposal, *best* which is the best proposal selected by the initiator and *criteria* which is a set of functions which help to sort the different proposals. The role requires a capacity which is named *ChooseBestBid*. The behavior of the initiator role specified by the behavior schema consists of three states. The first and by default state is *idle*. Whenever the *taskToBeDistributed* event occurs, it means that initiator will delegate a task, the initiator sends a call for proposal ($cfp(t)$ action) for a specific task t and enters the *waitingBids* state. In this state the initiator receives proposals and after a predefined timeout the initiator select among the bidders and send the corresponding answers. It then enters the waitingResult state waiting to receive a result from the chosen bidder. After the result is sent or a *timeout* has occurred the initiator returns to *idle* state.
The criterion used by an *Initiator* to choose a proposal are specified by a set of functions. Each function ranks with an integer a proposal as defined by the Criterion type. The *criteria* set is a set of such functions. It specifies a multi-criteria ranking for the proposals.

$$Criterion == f : Proposal \rightarrow \mathbb{N}$$

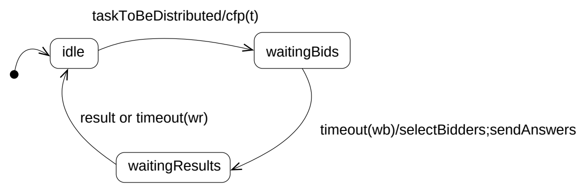
Initiator

Role

$proposals : \mathbb{F}\, Proposal$
$best : Proposal$
$criteria : \mathbb{F}\, Criterion$

$ChooseBestBid \in requiredCapacities$

behaviour


taskToBeDistributed/cfp(t)
idle
waitingBids
result or timeout(wr)
timeout(wb)/selectBidders;sendAnswers
waitingResults

selectBidders

$best' = capacityCall(ChooseBestBid(proposals, criterion))$

sendAnswers

The *ChooseBestBid* capacity inherits from the Capacity class. It *inputs* are a set of proposals and a set of criterion. It produces as *output* a proposal, which is the best according to the criteria, among the proposals in input. The proposals input set must not be empty in order to select one. It is the constraint stated in the *requires* set and the *ensures* set states that the best proposal is the best according to the *criteria* set.

ChooseBestBid

Capacity

$inputs = \mathbb{F}\, Proposal, \mathbb{F}\, Criterion$
$outputs = \{best : Proposal\}$
$requires : \{inputs \neq \varnothing\}$
$ensures : \nexists bid : inputs \bullet$
$\quad \exists\, i \in [1..\#criteria] \bullet$
$\quad criterion(i, bid) < criterion(i, best)$
$\quad \wedge\, \forall\, j \in [1..\#criteria] \setminus i \bullet criteria(j, bid) = criteria(j, best)$

The *ChooseBestBidImpl* class specifies a possible implementation of the *ChooseBestBid* capacity. It inherits from *CapacityImpl* and has two attributes a *proposal* which is the selected best proposal and *bids* which is a set of proposals. The behaviour schema specifies that at first the best proposal is initialized by the first proposal and after that each proposal is compared in sequence with the best found. If it is better than the current best according to the *min* operation it becomes the new best and the capacity implementation iterates through the bids sequence. The *min* operation returns the best proposal among two proposals. *head* and *tail* operations return respectively the first and the rest of the proposals sequence.

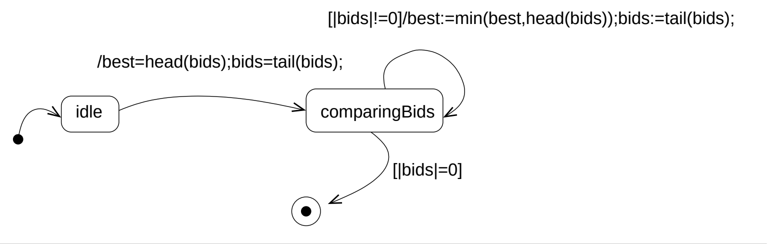
ChooseBestBidImpl

CapacityImpl

$best : Proposal$
$bids : \mathbb{F}\, Proposal$

$implements = \{ChooseBestBid\}$

behaviour


[|bids|!=0]/best:=min(best,head(bids));bids:=tail(bids);
/best=head(bids);bids=tail(bids);
idle
comparingBids
[|bids|=0]

min

$prop_1?, prop_2?, res! Proposal$

$prop_1? \prec prop_2? \Rightarrow res! = prop_1?$
$prop_2? \precsim prop_1? \Rightarrow res! = prop_2?$

head

$sequence? : \operatorname{seq} Proposal$
$res! : Proposal$

$res! = sequence?(1)$

tail

$sequence?, res! : \operatorname{seq} Proposal$

$sequence? = head(sequence?) \cup res!$

4.2 Verification

The specification of the contract net example was given as input to the SAL environment [7] which is a suite of model checkers and theorem provers. It was compared with the same specification but without the capacity concept. It means that the initiator role integrates the behaviour that choose the best proposal. The SAL environment integrates a path finder which generates traces from the semantics of the specification. The basic behaviour is to generate a ten steps trace of the system. The table 1 sums up the time in seconds taken by the different computations. The first line corresponds to the construction of the structure used by the path-finder and the second line is the generation of a ten steps trace. One can see that, even on the simple example described in this paper, the version with capacity is more efficient than the version without capacity. Indeed, the version without abstraction is more than four times longer than the one with capacity.

	Version with abstraction	Version without abstraction
Construction of the structure	0.15	0.63
Trace generation	1	0.23

Table 1. Execution time comparison

We were able to verify the following property for the *ChooseBestBidImpl* class.

$$\begin{array}{l} \nexists \mathit{bid}: \mathit{inputs} \bullet \\ \quad \exists\, i \in [1..\#\mathit{criteria}] \bullet \\ \quad \mathit{criterion}(i, \mathit{bid}) < \mathit{criterion}(i, \mathit{best}) \\ \quad \wedge\, \forall\, j \in [1..\#\mathit{criteria}] \setminus i \bullet \mathit{criteria}(j, \mathit{bid}) = \mathit{criteria}(j, \mathit{best}) \end{array}$$

This property corresponds to the *ensures* set of the *ChooseBestBid* capacity. In order to verify this property we have used a k-induction scheme as described in [8]. It means that we have to prove that the property holds for initial states and is preserved under each transition. The SAL bounded model checker associated with induction proved this property. The *ChooseBestBidImpl* capacity implementation verifies then the *ChooseBestBid* capacity.
Concerning the specification of the *Initiator* role with the *ChooseBestBid* capacity we have proven the following property using the symbolic model checker.

$$\mathit{behavior.state} = \mathit{waitingBids} \Rightarrow \Diamond(\mathit{behavior.state} = \mathit{idle})$$

We were then able to prove that the given specification satisfies the two properties that an *Initiator* always return to the idle state and the chosen proposal is allways the best one.

5 Related works and Conclusion

In this paper we have presented a framework of organizational concepts with a formal semantics which allow the use of abstraction during proofs. The abstraction is based on the capacity concept which abstracts a role know-how. The description of the capacity enables the abstraction of this know-how from the real implementations. The proofs of properties at the organization level are then less complex. This approach enables one to tackle the limitation of formal methods concerning the complexity of verification. These claims are illustrated through the contract net protocol specification example.

Formal methods have been widely used in the MAS field see [1] for a short survey. Among these approaches many use modal logics to specify and make proofs about MAS. Proofs using modal logics theories can be non trivial. In [15] a compositional approach is used for the verification of MAS. Compositional approaches are based on the following principle : if each component behaves correctly in isolation, then it behaves correctly in concert with other components. One has thus to prove each component and then the composition relationship in order to prove properties concerning the whole system. We have used an organizational framework which seems appropriate for MAS and HMAS modelling. For the future we plan to build a software environment which will help the specifier in his tasks of building and verifying specifications. Moreover, we plan to integrate this formal verification approach within the ASPECS methodological process [5, 6] which enables the analysis and design of MAS and HMAS.

References

1. Carole Bernon, Massimo Cossentino, and Juan Pavón. An overview of current trends in european AOSE research. *Informatica (Slovenia)*, 29(4):379–390, 2005.
2. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
3. Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. In *POPL*, pages 342–354, 1992.
4. M. Cossentino. *From Requirements to Code with the PASSI Methodology.* Idea Group Inc., Hershey, PA, USA., 2005.
5. Massimo Cossentino, Nicolas Gaud, Stéphane Galland, Vincent Hilaire, and Abderrafiâa Koukam. A holonic metamodel for agent-oriented analysis and design. In *HoloMAS'07*, 2007.
6. Massimo Cossentino, Nicolas Gaud, Stéphane Galland, Vincent Hilaire, and Abderrafiâa Koukam. A metamodel and implementation platform for holonic multi-agent systems. In *EUMAS'07*, 2007.
7. Leonardo de Moura, Sam Owre, Harald Rueß, John Rushby, N. Shankar, Maria Sorea, and Ashish Tiwari. SAL 2. In Rajeev Alur and Doron Peled, editors, *Computer-Aided Verification, CAV 2004*, volume 3114 of *Lecture Notes in Computer Science*, pages 496–500, Boston, MA, July 2004. Springer-Verlag.
8. Leonardo Mendonça de Moura, Harald Rueß, and Maria Sorea. Bounded model checking and induction: From refutation to verification (extended abstract, category A). In Warren A. Hunt Jr. and Fabio Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 14–26. Springer, 2003.
9. V. Dignum and F. Dignum. Coordinating tasks in agent organizations. or: Can we ask you to read this paper? In *Coordination, Organization, Institutions and Norms Engineering Societies in the Agents' World*, 2006.
10. Roger Duke, Paul King, Gordon Rose, and Graeme Smith. The Object-Z specification language. Technical report, Software Verification Research Center, Departement of Computer Science, University of Queensland, AUSTRALIA, 1991.
11. Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: an organizational view of multi-agent systems. In *Agent-Oriented Software Engineering IV 4th International Workshop, (AOSE-2003@AAMAS 2003)*, volume 2935 of *LNCS*, pages 214–230, Melbourne, Australia, July 2003.

12. FIPA. Fipa contract net interaction protocol specification. Technical report, FIPA, 2000.
13. Pablo Gruer, Vincent Hilaire, Abder Koukam, and P. Rovarini. Heterogeneous formal specification based on object-z and statecharts: semantics and verification. *Journal of Systems and Software*, 70(1-2):95–105, 2004.
14. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
15. Catholijn M. Jonker and Jan Treur. Compositional verification of multi-agent systems: A formal analysis of pro-activeness and reactiveness. *Int. J. Cooperative Inf. Syst*, 11(1-2):51–91, 2002.
16. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design: An International Journal*, 6(1):11–44, January 1995.
17. Michael Luck and Mark d'Inverno. A formal framework for agency and autonomy. In Victor Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260. AAAI Press, 1995.
18. Sebastian Rodriguez, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abder Koukam. An analysis and design concept for self-organization in holonic mas. In S Brueckner, S Hassas, M Jelasity, and D Yamins, editors, *Engineering Self-Organising Systems*, number 4335 in LNAI, pages 15–27. 2007.
19. R. G. Smith. The contract net protocol : High-level communication and control in a distributed problem solver. *Morgan Kaufmann*, pages 357–366, 1988.
20. F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the gaia methodology, 2003.