Requirements for achieving software agents autonomy and defining their responsibility

GOUAICH Abdelkader[1]

Laboratoire Informatique, Robotique et Micro Electronic- UMR 5506 -
161, rue Ada - 34090 Montpellier cedex 5 - France
gouaich@lirmm.fr

Abstract. This paper addresses the problem of implementing agent-based software systems with respect to agent framework fundamental concepts such as autonomy and interaction without specifying any particular agent internal architecture. The autonomy and interaction axioms imply that a deployment environment has to be defined in order to achieve interaction among agents. This deployment environment may also encode environmental rules and norms of the agent society. The responsibility of an agent is then defined as being in adequacy with its environmental rules. Finally, a formal deployment environment, named MIC*, is presented with a simple application showing how interaction protocols are guaranteed by the deployment environment, which protects agents from non-conform actions and preserve their autonomy.

1 Introduction

Agent-based software engineering seems to be an interesting approach for the design and development of open and distributed software systems [Fer95,JW97]. According to Demazeau [Dem95] a multi-agent system can be described with four main concepts: Agent, Interaction, Organisation and Environment. Agents are autonomous goal-directed entities that populate the multi-agent system. They achieve their design goals by interacting with other agents and using resources available in their environment; interaction is defined by Ferber [Fer95] as the set of mechanisms used by agents to share knowledge or to coordinate activities. Organisation in multi-agent systems can be defined as the set of mechanisms that reduces the entropy of the system and makes it ordered, behaving as a coherent whole. The environment represents a container where agents live. According to Russell and Norvig [RN95] the environment is one of the fundamental concepts in programming and designing agent-based systems. However, there is still some confusion on the definition of this concept. In fact, situated agents works such as agent-based simulation, artificial life or robotics consider the environment as the space where agents are situated; perceive their vicinity; and access to the available resources. On the other hand, agent-based software engineering considers the agent environment as the software components surrounding agents and offering them some computing facilities. To avoid confusion, the later definition will be referenced as agent deployment environment. The presented work is concerned with the study of a generic formal deployment environment and its properties to implement autonomous agent-based software system.

1.1 Paper outlines:

This paper is organised as follows: starting from agent features, section 2 identifies the necessary requirements to implement autonomous agents. Section 3 studies responsibility of an autonomous agent as being coherent to its deployment environment laws. Section 4 presents an example of a formal deployment environment fulfilling the necessary requirements for agent's autonomy. In order to illustrate the concepts of this paper, a simple experiment is given in section 5 followed by the conclusion presented in section 6.

2 Requirements to implement autonomous agents

This section uses Wooldridge and Jennings agent's definition stating that [WJ95]:
"Perhaps the most general way in which the term agent is used is to denote a hardware or (more usually) software-based computer system that enjoys the following properties:

- *autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state [Cas95];*
- *social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language [GK94];*
- *reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;*
- *pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative."*

This definition specifies some features that a physical or software entity must fulfil to be considered as an agent. Still, it does not specify how to build software agents. Consequently, developers that are in charge of implementing agent-based systems might have their own interpretation of these agent's features. For instance, autonomy is usually interpreted as concurrency. So, a simple concurrent object implements autonomous agents. These misinterpretations of agent fundamental features have led several software engineering communities to consider agent-based systems just as a renaming of concurrent object systems. This section studies carefully agent autonomy and social ability features in order to derive the requirements on the software system implementing the agent-based system. This axiomatic approach proves that a non-agent entity is needed in order to achieve interaction among autonomous agents.

2.1 What is an autonomous agent?

Autonomy is given several interpretations. Two main interpretations were selected for the purpose of this paper: autonomy as self-governance and as independence.

Autonomy as self-governance: Steels in [Ste95] tries to understand the difference between a standard software program and an autonomous agent. His paper refers to a personal communication with Tim Smithers (1992) on autonomy:
"The central idea in the concept of autonomy is identified in the etymology of the term: autos (self) and nomos (rule or law). It was first applied to the Greek city-state whose citizens made their own laws, as opposed to living according to those of an external governing power. It is useful to contrast autonomy with the concept of automatic systems. The meaning of automatic comes from the etymology of the term cybernetic, which derives from the Greek for self-steering. In other words, automatic systems are self-regulating, but they do not make the laws that their regulatory activities seek to satisfy. These are given to them, or built into them. They steer themselves along a given path, correcting and compensating for the effects of external perturbation and disturbances as they go. Autonomous systems, on the other hand are systems that develop, for themselves, the laws and strategies according to which they regulate their behaviour: they are self-governing as well as self-regulating. They determine the paths they follow as well as steer along them"
This description defines autonomy of a system (or entity) as its ability to produce its own laws and to follow them. In contrast, an automatic entity is given what to do and follows these instructions without making new laws or changing them. This notion of autonomy is shared by Castelfranchi [Cas95] who defines an autonomous agent as a software program able to excise a choice that is relevant in the context of goals-directed behaviour. Luck and D'iverno in several works [Ld95,LD01,dL96] share also the same notion of autonomy and specify formally using Z notations [Spi87] what is an autonomous agent and what distinguish it from an object or a simple agent. From Luck and D'iverno perspective, an object is a simple software entity with some features and actions. An agent is an object increased with a set of goals to be achieved. Autonomous agent is then defined as an agent with a set of motivations steering the agent in selecting what goals have to be achieved. Guessoum and Briot [GB99] addresses also this issue and enrich a pure object communicating architecture, Actalk, with some mechanisms such as controlling message reception and selecting what behaviour to adopt, in order to build an agent platform named DIMA.

Autonomy as independence: Social dependence network (SDN) has been introduced by Sichman et al. in [SCCD94] to allow social agents to reason about and understand their artificial society in order to achieve their goals. Within this framework, agents have external descriptions representing models about their neighbours. An external description of an agent is composed by its goals, its actions, its resources and its plans. Goals represent the state of affair that the agent want to reach; actions are operations that an agent is able to perform; resources represents the resources that an agent has control on; and finally plans are sequences of actions and resources. Notice that the authors adopt hypothesis of external description compatibility implying that all agents have the same models about others. This is unachievable in an open context where the set of agents is not static and the interaction is asynchronous. SDN framework distinguishes three forms of autonomy. An agent is a-autonomous for a given goal according to a set of plans, if there is a plan in this set that achieves the goal, and every action in each

plan belongs to its capabilities. An agent is considered as r-autonomous for a given goal according to a set of plans, if there is a plan in this set that achieves the goal, and every resource in each plan belongs to its resources. Finally, an agent is s-autonomous when it is both a-autonomous and r-autonomous. According to this definition, an agent is autonomous for a particular goal if it does not depend for resources or actions on another agent. These definitions are then used to define dependency relation among agents. Following the SDN definitions, autonomy is easily proved as equivalent to the independence. Hence, agents are autonomous if and only if they are independent from all other agents.

Discussion: Sichman et al. define autonomy of agents as being independent on actions and resources from other agents. On the other hand, [Cas95,Ste95,Ld95] define agents autonomy as its ability to reason about, generate and execute its goals by its own. Our interpretation of agent autonomy is closer to autonomy as self-governance than autonomy as independence. In fact, as mentioned by [Ld95], a pocket calculator that has the resources and actions to perform to calculate some arithmetical operation is seen as autonomous agent according to SDN definitions. Our interpretation on autonomy relates more on the decisional process than on dependencies on resources or knowledge. In fact, a software agent is an autonomous entity if it behaves differently depending on its internal context. In other words, if agents have to be modelled as mathematical functions, the observer of the agent has to admit that he/she is not able to know all the parameters of this function. Consequently, this observer will never have the complete knowledge to predict the autonomous agent reaction and behaviour.
According to this particular interpretation of autonomy, the problem now is to know how to correctly implement autonomous agents. In other words are usual techniques such as concurrent object sufficient to implement autonomous agents? This question has been answered partially by [Ld95] with the concept of motivation. Hence, motivated-agents generate their goals by considering their current motivations. Nevertheless, introducing motivations in an agent shifts the problem from how to manage agent's goals, to how to manage agent's motivations. In fact, any external entity that has a full control of the software structure representing the agent's motivations controls the behaviour of this agent. Guessoum and Briot's approach seems also interesting to implement autonomous agents. However, as it is the case for motivated-agent, it specifies precisely what should be the agent internal architecture in order to guarantee its autonomy. Our goal is to study the minimal necessary requirements of agent-based software architecture without considering any particular internal architecture of agents.

Internal integrity to guarantee agent's autonomy: Integrity of the agent software structure is a sine qua none condition to implement autonomous agents without considering any particular agent internal architecture. The term agent software structure refers to the set of data and instructions that encode the agent and define its computational behaviours. In fact, if this structure is accessible and modifiable by another agent, the decisional process and behaviours may be altered. For instance, if the set of motivations of motivated-agent is accessible by another entity, the agent loses its autonomy. Sim-

ilarly, if the decisional process is accessible by an external entity, DIMA autonomous agents lose their autonomy. Consequently, an autonomous agent should not allow any external agent to change its software structure either by setting a feature to certain value or by calling a side-effect method. On the other hand, agents are interacting entities. So, agents change the perceptions of other agents. Since, the perceptions of agents are part of their software structure, this contradicts the autonomy statement. To avoid this contradiction, the first solution is to consider that the software structure representing the agent's perception (or its inbox) does not belong to its software structure. This is not achievable in practice. In fact, software engineering requirements on modularity include the agent's perception in the agent's software structure. The other approach is to define a non-agent entity that is in charge of achieving the interaction among agents. This entity is what was identified as agent deployment environment. The deployment environment is not an autonomous agent and is considered as an automatic software system. Hence, its software structure and rules are defined once and should be followed by agents. This does not mean that dynamic deployment environment cannot be constructed. However, a dynamic deployment environment is not as single entity, but a sequence of different entities. This result is particularly interesting to establish agents' responsibility in an open and untrusted software environment. In fact, without this feature the responsibility of agents when they violate the multi-agent society norms cannot be established. This point is discussed in more details in the next section.

3 Agent responsibility in open software systems

Biological and physical environments as examples of multi-agents systems that are separated in two dimensions: the agents and the physical environment. Biological agents are subject to environmental laws and principles. Agents do not modify the environmental laws; they have to deal with them in order to achieve their goals. For instance, if a human agent's goal is to fly, he/she will never modify the physics law on gravity, but use other physics laws on aerodynamics in order to achieve its goals. This analogy can be used in order to build secured agent-based systems. In fact, the design requirements of a software system are the environmental laws. Any deployed known or unknown agent has to deal with these environmental laws. The term unknown agent refers to software agent whose software structure is completely unknown. This may be the case of agents that are designed and implemented by external organisations. In contrast, a known agent is a software agent whose software structure is accessible (open source or developed locally). In large open software systems, agents have to be assumed as unknown. Thus, the deployment environment has to identify agents that challenge its internal laws. This defines agent's responsibility as being coherent to the deployment environment rules and laws. To guarantee these points, agents are not allowed to change directly their environment. Their actions have to be discrete and explicitly represented as attempts of actions. The deployment environment considers how to react to these attempts. Attempts that violate the environmental laws are simply ignored. The emitting agent is then considered as responsible for violating the environmental laws. Furthermore, agents cannot deny this responsibility since the deployment environment is an automatic system that does not modify its internal laws. In fact, if this was not the case,

agents may claim that they cannot conform rules since they are arbitrarily changed. For instance, interaction protocols are examples of agent society rules that have to be followed by agents in an open agent-based system. It would be interesting to generate a deployment environment that encodes and guarantees these interaction norms without specifying any particular agent internal architecture. This is considered as a case study and studied in section 5.

4 A formal agent deployment environment

This section presents an example of a formal deployment environment, named MIC* for $\{$Movement, Interaction, Computation$\}^*$ [GGM03], fulfilling the requirements presented before on agent autonomy and responsibility. MIC* is an abstract structure where autonomous, mobile and interacting entities are deployed. Within this framework, all interactions are conducted by explicitly exchanging interaction objects through interaction spaces. Hence, agents do not alter directly the deployment environment or the perceptions of other agents, but send their attempts as interaction objects. Interaction objects are structured: in fact, a formal addition law can compose them commutatively $+$ to represent simultaneous interactions. Furthermore, abstract empty interaction object 0 can be defined to represent no interaction. The less intuitive part of the structure of the interaction objects concerns negative interaction objects. Negative interaction objects are constructed formally and may have no interpretation in the real world. However, they are useful for the internal model definitions and implementation of the deployment environment. For instance, the deployment environment can cancel any action, x, of the agent simply by performing an algebraic operation, $x + (-x) = 0$, that is expressed within the model notations. Finally, interaction objects defines a structure of a commutative group $(\mathcal{O}, +)$, where $\mathcal{O}$ represents the set of interaction objects and $+$ the composition law. Interaction spaces, represented by $\mathcal{S}$, are defined as abstract locations where interaction between agents holds. They are active entities that control their local and specific interaction rules. For instance, interaction object that are sent inside an interaction space may be altered if they violate the interaction norm. Agents, represented by $\mathcal{A}$, are autonomous entities that perceive interaction objects and react to them by sending other interaction objects. As said before, agents' actions are always considered as attempts to influence the deployment environment structure. These attempts are committed only when they are coherent with the deployment environmental rules of evolution.
Having these elementary definitions, each MIC* term is represented by the following matrices:

- Outboxes Matrix: The rows of this matrix represent agents $A_i \in \mathcal{A}$ and the columns represent the interaction spaces $S_j \in \mathcal{S}$. Each element of the matrix $o_{(i,j)} \in \mathcal{O}$ is the representation of the agent A_i in the interaction space S_j.
- Inboxes Matrix: The rows of this matrix represent agents $A_i \in \mathcal{A}$ and the columns represent the interaction spaces $S_j \in \mathcal{S}$. Each element of the matrix $o_{(i,j)} \in \mathcal{O}$ defines how the agent A_i perceives the universe in the interaction space S_j.
- Memories vector: Agents $A_i \in \mathcal{A}$ represent the rows of the vector. Each element m_i is an abstraction of the internal memory of the agent A_i. Except the existence

of such element that is proved using the Turing machine model, no further assumptions are made in MIC* about the internal architecture of the agent.

4.1 Structure dynamics

The previous part has presented the static objects to fully describe environmental situations or states. In this section, three main evolutions of this static description are characterised:

- Movement μ: A movement is a transformation μ, of the environment where both inboxes and memories matrices are unchanged, and where outboxes matrix interaction objects are changed but globally invariant. This means that the interaction objects of an agent can change positions in the outboxes matrix and no interaction object is created or lost.
- Interaction ϕ: The interaction is characterised by a transformation ϕ that leaves both outboxes and memories matrices unchanged and transforms a row of the inboxes matrix. Thus, interaction is defined as modifying the perceptions of the entities in a particular interaction space.
- Computation γ: An observable computation of an entity transforms its representations in the outboxes matrix and the memories vector. For practical reasons, the inboxes of the calculating entity are reset to 0 after the computation to distinguish interaction objects that were involved in different computations.

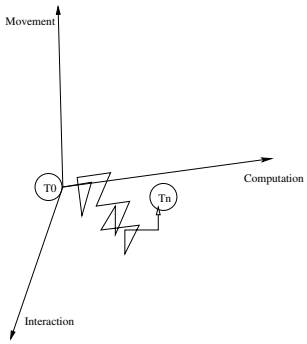

Fig. 1. Evolution of a MAS deployment environment starting from an initial term T_0 until T_n by following elementary transformations

These three elementary evolutions are orthogonal. For a particular agent-based system, the structure of MIC* is fully defined by having the interaction objects group; the sets of processes and interaction spaces; the sets of transformations μ, ϕ, γ and a combination order of these transformations. $\{\mu|\phi|\gamma\}*$ (MIC*) is the most general combination order, where $|$ expresses an exclusive choice between elements. The main idea of this approach is that the dynamics of the deployment environment can be modelled using the (μ, ϕ, γ)-base. Consequently, as presented in figure 1, any state of the deployment environment can be reached by following a path of evolutions of type μ, ϕ or γ.

5 Experiment

The experiment scenario simulates a virtual city where autonomous agents: user agent, bogus agent and service agent move and may interact. The mobility were used to show another feature of MIC* in modelling ubiquitous software systems. This is not addressed in this paper. The user agent goal is to buy a ticket from a ticket selling service. The goal of the bogus agent is to challenge the established norm on interaction. Hence, this agent follows the interaction protocol until a step n, where n is a random number. After this, the following communicative acts and actions are randomly sent.

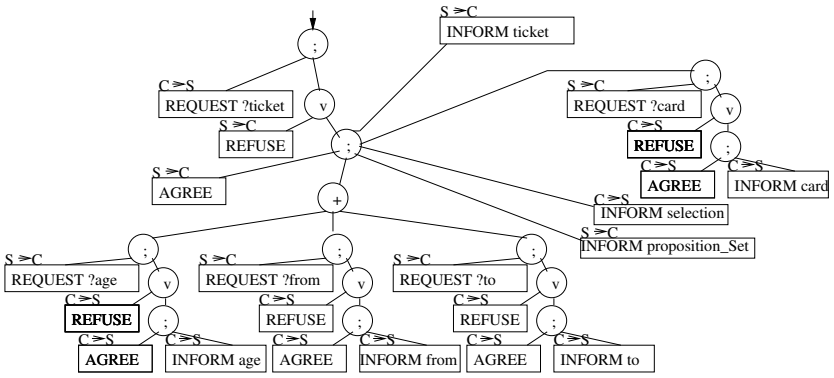

Fig. 2. Ticket buying interaction protocol between the user agent (C) and the service (S)

Ticket buying interaction protocol: In order to validate the presented approach, a simple interaction protocol has been defined in order to buy a ticket from the ticket selling service. The notations that are used to describe this interaction protocol are the following: $(a);(b)$ expresses a sequence between two interactions a and b. This interaction is valid if and only if a happens first followed by b. $(a) \vee (b)$ expresses an exclusive choice between a and b. This interaction is valid if and only if a or b hold. $(a) + (b)$ expresses parallel interactions that hold in an unordered manner. Hence, this interaction is valid if and only if a and b happen no matter which first. Having these notations the 'ticket-buying' interaction protocol is described in figure 2. The user agent (C) initiates this protocol by sending a request to the service (S). After this, the service agent may agree to respond to the request or not. When agreeing, the service agent asks the user agent some information about the starting point; the destination of his travel and the traveller's age. After gathering these data, the service agent delivers some propositions to the user agent. When the user agent selects an offer, he informs the service agent about it. After the payment procedure, the offer is acknowledged and the dialogue is closed. The interaction protocol is specified from the viewpoint of an external observer. Having this specification a MIC*-deployment environment is generated. This process is discussed more on details in [GOU03]. All autonomous agents are executed as concur-

rent processes that interact with the deployment environment with explicit interaction objects sent via low-level TCP/IP sockets.

5.1 Results

By following the interaction protocol, the client interacts with the service agent and succeeds in buying a ticket. In contrast, attempts of the bogus agent have never reached the perceptions of the service agent. In fact, the deployment environment checks the validity of the dialogue among agents by recognising valid sequences of messages [GOU03]. When a sequence of messages is invalid according to the specification of the interaction protocol, the interaction attempt is simply ignored. Formally, this is expressed within MIC* as an evolution with the identity application. So, the deployment environment is not modified. Consequently, the service agent is protected from erroneous interactions. This can be considered as raising the autonomy of the agent. In fact, handling erroneous interactions is an agent's behaviour that may affect other application-level behaviours by consuming agent's resources. Furthermore, when error cases are not treated correctly the whole agent activity may be altered. On the other side, without any assumption on the bogus agent's software structure, the deployment environment was able to establish its responsibility for being not compliant with the interaction norms. Finally, since the internal software structure of agents was not accessible, their behaviours and decisional process have been conserved, which was considered in this paper as a sine qua none condition in order to achieve agents' autonomy.

6 Conclusion

This paper has addressed the problem of implementing agent-based software systems with respect to agent framework fundamental concepts such as autonomy and social abilities. This works have been conducted without specifying any particular internal agent architecture. Guaranteeing the internal integrity of the agent software structure has been found as a necessary condition to correctly implement autonomous agents. Hence, autonomous agents are completely closed boxes that perceive and react to interaction objects coming from their external world. Consequently, to conduct interaction among these autonomous entities, a non-agent entity is necessary. This entity was defined as the deployment environment and had been used in order to encode some agent society laws and norms. When encoding these norms, the deployment environment establishes a referential determining the responsibility of each autonomous agent in an open context. Finally, an example of deployment environment fulfilling the requirements on agent autonomy and responsibility has been presented. Finally, a simple application has shown, how a agent-based system has been implemented to simulate economical agents. The deployment environment has encoded the interaction norms and guaranteed that illegal interactions do not disturb other agents. The autonomy and safety of agents has been conserved.
The next step of our works is to link social models describing an agent-based system to the MIC* model. Hence, starting from a semi-formal description of the multi-agent system, using AUML or another formalism, a specific deployment environment can be

automatically generated and defined independently from the autonomous agents that will populate it.

References

[Cas95] C. Castelfranchi. Guarantees for autonomy in cognitive agent architecture. *Intelligent Agents: Theories, Architectures, and Languages*, 890:56–70, 1995.

[Dem95] Yves Demazeau. From interaction to collective behaviour in agent-based systems. In *1st European Conference on Cogntive Science*, St Malot, France, 1995.

[dL96] M. d'Inverno and M. Luck. A formal view of social dependence networks. In Zhang and Lukose, editors, *Distributed Artificial Intelligence Architecture and Modelling: Proceedings of the First Australian Workshop on Distributed Artificial Intelligence*, pages 115–129. Springer-Verlag: Heidelberg, Germany, 1996.

[Fer95] Jaques Ferber. *Les Systemes Multi-Agents*. InterEditions, 1995.

[FIP96] FIPA. Foundation of intelligent and physical agents, 1996. http://www.fipa.org.

[GB99] Zahia Guessoum and Jean-Pierre Briot. From active objects to autonomous agents. *IEEE Concurrency*, 7(3):68–76, 1999.

[GGM03] Abdelkader GOUAICH, Yves GUIRAUD, and Fabien MICHEL. Mic*: An agent formal environment. To appear in the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003), 7 2003. Orlando, USA.

[GK94] M.R. Genesereth and S.P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 1994.

[GOU03] Abdelkader GOUAICH. Implementing interaction norms is open, distributed and disconnected multi-agent software systems. Submitted to AOSE 2003. Url: http://www.lirmm.fr/~gouaich/research.html, 2003.

[JW97] Nicholas R. Jennings and Michael Wooldridge. Agent-based software engineering. In *Software Engineering*, volume 144, pages 26–37. IEEE, 1997.

[Ld95] 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 (ICMAS-95)*, pages 254–260, San Francisco, CA, USA, 1995. AAAI Press.

[LD01] M. Luck and M. D'Inverno. Autonomy: A nice idea in theory. *Intelligent Agents VII: Proceedings of the Seventh International Workshop on Agent Theories, Architectures and Languages*, 2001.

[OPFB02] James J. Odell, H. Van Dyke Parunak, Mitch Fleischer, and Sven Brueckner. Modeling agents and their environment. In *AOSE 2002, AAMAS 2002*, Bologna, 2002.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[SCCD94] Jaime Simão Sichman, Rosaria Conte, Cristiano Castelfranchi, and Yves Demazeau. A social reasoning mechanism based on dependence networks. In A. G. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence*, pages 188–192, Chichester, 8–12 1994. John Wiley & Sons.

[Spi87] J.M. Spivey. *The Z notation A reference manual*. Prentice Hall, 1987.

[Ste95] Luc Steels. When are robots intelligent autonomous agents? *Robotics and Autonomous Systems*, 15:3–9, 1995.

[WJ95] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.